言語と構文解析

徳田雄洋 著

情報数学講座

5

共立出版株式会社

言語と構文解析

徳田雄洋 著

情報数学講座

5

共立出版株式会社

刊 行 主 旨

情報科学に使われている数学は，非常に幅広く，また奥深い．さらには，情報科学から発展して数学の一部門を形成しつつある分野もある．そのため，現在の大学教育の中で「情報科学のための数学」を体系化することは，まだ十分にできているとはいい難い．それは，数学のさまざまな分野から情報科学に役立つものを摘出し，限られた期間内に効果的に学生に提供することの困難さと，情報科学が新しい学問であり流動的であるだけに，その中で役立つ数学を整理し統合化しようというには，まだ時期尚早なためでもある．しかし，情報科学がしっかりと地につき発展していくためには，どの時点においても数学との協調が必要不可欠である．

このような視点のもとに，現時点での「情報科学のための数学」を大胆に整理・統合化し，本「情報数学講座（全15巻）」を構成した．そのため，ある分野は重点的にとりあげ，ある分野は縮小・併合することになった．たとえば，数値解析が省かれているが，これはそれだけで一つのシリーズを形成するほど大きな分野であることもあって，本講座では他にゆだねることにした．

各巻の内容については，次のことを目標に定めた．

① 「役に立つ数学」を提供することをモットーとする．

② 定理・証明の羅列は控え，話題の展開や表現の仕方を工夫する．

③ コンピュータ・サイエンスからの具体例を豊富に入れ，抽象的概念をわかりやすく解説する．

④ 「その彼方に何があるのか」，「その分野が何を目指しているのか」に

言及し，筆者のフィロソフィを重視している．

本講座が情報科学の発展の一助になれば幸いである．

編集委員

まえがき

「人間の子供は，生まれてから3年間程度，周囲の人間の話す有限個の文を聞いていると，やがて無限個の文が作れるようになるのは何故だろうか？」という自然な疑問から出発した1950年代の形式言語理論は，その後，プログラミング言語処理と自然言語処理の共通の基礎理論に成長し，さらに応用技術としてコンパイラ生成系や構造エディタ生成系のための方法論に発展して行った．

本書は，大学学部教育の範囲で，プログラミング言語処理と自然言語処理の共通の基礎理論にあたる形式言語とオートマトンの理論を述べ，さらに応用技術として字句解析法，構文解析法，意味評価法のアルゴリズムについて体系的に説明した教科書である．

本書の特徴は次の通りである．

(1) 通常は別々の授業で扱われる，形式言語理論の話題とコンパイラ構成の話題の共通部分に対し，本書一冊で統一的な見通しを得られるように構成を配慮した．
(2) 1950年代に開発された古典的な方法から1990年代に開発された最近の方法までの関連アルゴリズムの中から，理解しやすくかつ基本的と考えられるアルゴリズムを収録した．
(3) 形式言語の基礎理論を，単なる数学的理論としてとらえるのではなく，ソフトウェアを自動的に生成するための基礎的方法論としてとらえることを強調した．

本書の内容は，主として著者が東京工業大学工学部で数年間にわたって行った授業に基づいている．また非常勤講師として，それぞれ著者が半年間行った東京大学理学部の授業および東京工科大学の授業の内容も参考とした．

本書の執筆にあたり，多くの方々からさまざまな御助言，御協力を頂いた．特に，編集委員の小林孝次郎先生と鈴木則久先生には，原稿に対し貴重なコメントを頂いた．感謝の意を表したい．原稿の最終調整の段階で，東京工業大学大学院情報理工学研究科計算工学専攻徳田研究室の学生の方々のさまざまな協力を得た．特に，学部学生の松崎智広君，矢野浩邦君，大学院生の鈴木徹也君，山本晃治君，木村喜昌君の原稿の内容についての有益なコメントに対し，謝意を表したい．最後に東京工業大学大学院情報理工学研究科計算工学専攻の渡辺喜道助手には，本書の内容全体の技術的チェックから紙面制作の各行程まで，多くの御協力を頂いた．深く感謝の意を表したい．

本書の内容には技術的にかなり深い内容も一部含まれているが，若い研究者，技術者，学部学生の方々のソフトウェアの基礎に対する理解を深める参考の書となれば，幸いである．

1995 年 9 月

徳田　雄洋

目　　次

第1章　言 語 処 理

この章では，本書のテーマである言語処理の目標，言語の研究方法，言語の構成要素を概観し，次に言語の構文定義と意味定義について一般的に述べ，最後に言語処理の例を示す．

1.1　言語処理の目標

言語には，日本語，英語などのように日常生活の一般的コミュニケーションのために人間社会とともに歴史的に発達してきた**自然言語** (natural language) と，特定目的のコミュニケーションのために人為的に制作され使用されてきた**人工言語** (artificial language) がある．人工言語のうち，特に人間とコンピュータのコミュニケーションのために作られた言語が，Pascal や C などの**プログラミング言語** (programming language) である．

プログラミング言語も含め，自然言語や人工言語に共通して行われる典型的な情報処理の形態に，**翻訳** (translation) と**解釈** (interpretation) がある．翻訳は，1つの言語で記述された表現から，意味を保ちながら別の言語の表現に，書き直すことである．これを日常生活の例で述べるならば，英語で書かれたハンバーガーの作成指示書を，英日翻訳して，日本語のハンバーガーの作成指示書を作る作業に相当する．つまり，言語表現の変換が主たる仕事であり，実際にハンバーガーを作る必要はないのである．

これに対し解釈は，1つの言語で記述された表現をそのまま読解し実行す

る作業である．先のハンバーガーの作成指示書の例で述べるならば，英語で書かれたハンバーガーの作成指示書を読みながら，実際にハンバーガー自体を作ってしまうことに相当する．すなわち，解釈では言語表現の意味を読み取り，指示通り実行することが主たる仕事となる．

例 1.1 (翻訳と解釈)
算術式の場合で，翻訳と解釈の相違を示そう．以下のような式

A:=2+3×5

に対し，翻訳の作業を行うと，例えば次のような命令列を，翻訳結果として出力することになる．

定数 3 と定数 5 を掛け算し，結果を変数 T0 へ格納せよ． 定数 2 と変数 T0 の値を足し算し，結果を変数 T1 へ格納せよ． 変数 T1 の値を，変数 A へ格納せよ．

同一の式に対し，解釈の作業を行うと，直接に指示通り計算を行うので，次のような動作の実行過程となる．

定数 3 と定数 5 を掛け算し，結果 15 を得る．
定数 2 と 15 を足し算し，結果 17 を得る．
17 を変数 A に格納する．

この結果，変数 A は値 17 を最終的に保持することになる．　□

1.2　言語の研究方法

言語の研究方法は，伝統的分類法に従うと，次の 3 つの方法に分けることができる．これらをそれぞれ**構文論** (syntax)，**意味論** (semantics)，**語用論** (pragmatics) と呼ぶ．

1. **構文論**
   構文論では，文を構成する記号の単位や，記号を構成する原子的記号の単位を定義する．そして，これらの文を構成する記号の並び方の規則性

を調べ，正しい記号の並び方を定義する．それぞれの記号や文全体の具体的意味は，構文論では特に問題としない．

2. **意味論**

意味論では，構文の正しい文に対して，各記号の意味や文全体の意味を定義する．意味を定めるための規則は，自然言語の場合，自然言語自体を用いて定義することが多い．プログラミング言語の場合は，対応する機械語の提示や形式的意味論を用いて定義する．

3. **語用論**

語用論では，実際に使用された特定の文の意味を，意味論が定める意味だけからでなく，さらにその文の送り手，受け手，環境，場所，時刻に関する知識を利用して，解釈・定義する．主として自然言語の場合に行われる．語用論では，特定の話者によって発言，記述された文の意味が，解析の対象となる．

例 1.2 (構文論・意味論・語用論)

「この文は英語の文である.」を例にして3つの研究方法を比べてみよう．構文論の立場では，文の構成要素の区切り方や，記号の並べ方が問題となる．したがって,「この文は英語の文である.」を,「この」「文」「は」「英語」「の」「文」「で」「ある」「.」というように分解し，これらの単語や句読点の並べ方が正しいかどうかを問題とする．幸い「この文は英語の文である.」は正しい単語の並べ方である．次に意味論の立場では,「この文は英語の文である.」の意味するものを検討することになる．この場合，意味するところのものも幸い比較的に単純で,「ここに指し示す記号の列は，米国や英国などで使われている言語である英語によって書かれた記号列である.」ということを示している．しかしながら最後に，語用論の立場による意味の検討を行うと，事情は少々複雑となる.「この文」と呼んでいるところのものが「この文は英語の文である.」自身である場合，奇妙な文となってしまうからである．つまり「この文は英語の文である.」という文自身は，日本語の文で英語の文ではないのである．□

1.3　言語の構成要素

言語の構成要素を，自然言語とプログラミング言語の場合で分析してみよう．自然言語の例として日本語の場合で考えると，構成要素の階層は大きいものから小さいものの順で次のようになる．

文章－文－文節－単語－形態素－文字

プログラミング言語の場合の構成要素の階層も，同じく大きいものから小さいものの順で次のようになる．

プログラム－宣言文・実行文－トークン－文字

それでは，本書で扱う範囲の構成要素の単位の定義を述べよう．

定義 1.1 (言語の構成要素)

1. 意志疎通の最小単位となる，記号列の並びを，**文** (sentence) と呼ぶ．文は，まとまった意味を持つ最小の単位と考えられる．
2. 文を構成する記号を，**字句記号** (token) と呼ぶ．字句記号は，1つの物体や動作や概念などを指し示す記号と考えられる．
3. 1つの字句記号は，それ以上は分割できない原子的な記号の1個以上の列によって構成される．この原子的な記号を，**文字** (character) と呼ぶ．文字の集合は，通常数十個から数千個程度の有限集合である．
4. 1つの基準に基づく文の集合を，**言語** (language) と呼ぶ．日本語，英語，プログラミング言語 Pascal，C は，この意味において，いずれも言語である．□

文字から字句を構成するための規則を，**字句規則**，字句の正しい並べ方の規則を，**構文規則**，正しい字句の並びの意味を定める規則を，**意味規則**と呼ぶ．

例 1.3 (言語の構成要素)

以下のような算術式の場合，

nagoya:=tokyo+osaka×120

変数名 nagoya，tokyo，osaka は，それぞれ 6 個，5 個，5 個の英字から作られ，数 120 は 3 個の数字から作られている．また演算記号 :=，+，× はそれぞれ 2 個，1 個，1 個の記号から作られている．□

1.4　構文定義と意味定義

言語処理では，その言語の構文と意味を適切に記述することにより，自動処理を可能とする．以下のような算術式を定義する例を取り上げ，

nagoya:=tokyo+osaka×120

構文と意味の記述の様子を示す．構文の記述としては，字句規則と構文規則の両方を定める必要がある．まず字句規則としては，この場合次の 3 種類の字句の表し方を定義する必要がある．

数の表し方，
変数の表し方，
演算子の表し方．

これらの定義は，次のように行うことができる．

（字句規則）

1. 数とは，数字 0 から 9 の長さ 1 以上の列である．
2. 変数とは，英字 a から z の長さ 1 の列に，英字 a から z，数字 0 から 9 の長さ 0 以上の列をつなげたものである．
3. 演算子とは，加算記号 +，減算記号 −，乗算記号 ×，除法記号 / の長さ 1 の列である．

構文規則としては次を記述しなければならない．

数と変数と演算子の並べ方

これらの定義は，次のように行うことができる．

（**構文規則**）

1. 数は式である．
2. 変数は式である．
3. 以下のように，式と演算子を並べたものは式である．

   式　演算子　式

意味規則としては，次を記述しなければならない．

　数と変数と演算子の並びが表す値の定め方

これらの値の定義は，次のように行うことができる．

（**意味規則**）

1. 式が数のみからなる場合，式の値はその数が表す値である．
2. 式が変数のみからなる場合，式の値はその変数が持つ数の値である．
3. 式が，以下のように式と演算子からなる場合，

   式1　演算子　式2

   式1の表す値と，式2の表す値に，演算子の表す演算を適用した結果が式全体の値である．

以上で数と変数と演算子からなる算術式の構文と意味のすべてが定義できたことになる．

1.5　言語処理の例

典型的な言語処理は対象言語の原子的記号の列が与えられた場合，次のような一連の作業を行うことである．作業3と作業4はそれぞれ先に述べた翻訳と解釈に相当する．

1. 与えられた原子的記号の列を，字句記号の列に変換する．この作業を，**字句解析** (lexical analysis) という．次のような原子的記号の列が与えられた場合，

   n a g o y a　: = t o k y o　+ o s a k a × 1 2 0 \n

   以下のように字句記号の列に切り分けることができる．

変数名	nagoya	加法演算子	+
区切り記号	空白	変数名	osaka
代入演算子	:=	乗法演算子	×
変数名	tokyo	整数定数	120
区切り記号	空白	改行記号	\n

結果として，区切り記号の空白を取り除き，次の字句記号の列を得る．

変数名 代入演算子 変数名 加法演算子 変数名 乗法演算子
整数定数 改行記号

2. 字句記号の列が与えられたとき，構文構造を決定し，構文論的に正しい文かどうかを判定する．この作業を，**構文解析** (syntax analysis) という．正しくない文の場合，構文論的な誤りを検出したり，訂正する．先の例では正しい文であるので，さらに，構文構造を次のように定めることができる．

(変数名 代入演算子 (変数名 加法演算子 (変数名 乗法演算子 整数定数)) 改行記号)

3. 正しい文の場合，別の言語の同じ意味の文に直す．例えば先に示した式は，次のような 4 つ組と呼ばれる別の言語表現に翻訳できる．

×	osaka	120	T1
+	tokyo	T1	T2
:=	T2		nagoya

4. 正しい文の場合，その意味を求めて，対応動作を行う．例えば，先の式の計算の実行を行うと，tokyo = 100，osaka = 5 のとき，nagoya = 700 となる．

第2章　字句規則と構文規則の記述

この章では，字句規則と構文規則の記述法について述べる．字句規則の記述には正規式，構文規則の記述には文脈自由文法をそれぞれ用いる．次に準備として，文脈自由文法の一般化にあたる句構造文法の例を示し，文脈自由文法上の計算，非決定性表現について述べる．

2.1　正　規　式

正規式は，字句記号の定義や探索記号列の定義などに広く用いられるパターン定義法である．まず，基本的概念である記号列についていくつかの用語を定義し，そして正規式を再帰的に定義する．

定義 2.1 (記号列)
重複も順序も区別する記号の並びを，**記号列** (string) という．0 個以上の n 個の記号 a_1，a_2，$\cdots$，a_n の並びを記法 $a_1a_2\cdots a_n$ で表す．並んでいる記号の個数 n を，記号列の**長さ** (length) という．長さ 0 の記号列を，**空列** (empty string) という．空列を記号 ε で表す．　□

記号の有限集合を，**アルファベット** (alphabet) という．アルファベット T の要素からなる有限の長さの記号列を，T の上の記号列という．T の上の記号列すべての集合を T$*$ で表す．T$*$ の部分集合を，**言語**という．L が T$*$ の言語のとき，集合 T$*$ − L を，言語 L の**補集合** (complement) という．

記号列 $a_1a_2\cdots a_n$ と記号列 $b_1b_2\cdots b_m$ は，$n=m$ で，しかも $a_1=b_1$，$\cdots$，$a_n=b_m$ のとき，**同一の記号列**であるという．列 $x=a_1a_2\cdots a_n$ に対し，i が 0 以上 n 以下のとき，記号列 $y=a_1a_2\cdots a_i$ を，x の**前部分列** (prefix) という．$i=0$ のときは空列である．同様に，列 $x=a_1a_2\cdots a_n$ に対し，i が 1 以上 $n+1$ 以下のとき，列 $z=a_ia_{i+1}\cdots a_n$ を，x の**後部分列** (sufix) という．$i=n+1$ のときは空列となる．

記号列 $w_1=a_1a_2\cdots a_n$ と記号列 $w_2=b_1b_2\cdots b_m$ が与えられたとき，w_1 と w_2 の**結合** (concatenation) $w_1\cdot w_2$ を $w_1\cdot w_2=a_1a_2\cdots a_nb_1b_2\cdots b_m$ と定義する．n を 1 以上とするとき，n 個の w を前から順番に結合した列 $(((w\cdot w)\cdot w)\cdot\cdots w)$ を記法 w^n で表す．任意の記号列 w について，

$$w\cdot\varepsilon=\varepsilon\cdot w=w$$

である．また任意の記号列 x，y，z について，

$$(x\cdot y)\cdot z=x\cdot(y\cdot z)$$

である．したがって，特に記号 a_1，a_2，$\cdots$，a_n を並べた記号列 $a_1a_2\cdots a_n$ が与えられたとき，どのような順序で結合が施されたと考えても，結果は同一の記号列となる．

定義 2.2 (正規式)

n 個の記号 (n は 1 以上の整数) の集合 $\{c_1,c_2,\cdots,c_n\}$ の上の**正規式** (regular expression) を次のように定義する．(ただし，この n 個の記号には記号 “ε”，“ϕ”，“+”，“・”，“*”，“(”，“)” は含まれないものとする.)

1. n 個の記号 c_1，c_2，$\cdots$，c_n はそれぞれ正規式である．これらはそれぞれ記号列の集合 $\{c_1\}$，$\{c_2\}$，$\cdots$，$\{c_n\}$ を表す．
2. 記号 ε は，正規式である．正規式 ε は，1 個の空列からなる集合 $\{\varepsilon\}$ を表す．
3. 記号 ϕ は，正規式である．正規式 ϕ は，空集合 $\{\}$ を表す．空集合は記号列を 1 つも要素に持たない集合である．
4. 正規式 e_1，e_2 に選択演算子 + を施した式

   $(e_1)+(e_2)$

は，正規式である．正規式 $(e_1)+(e_2)$ は，正規式 e_1 と e_2 の表す集合の和集合を表す．

5. 正規式 e_1，e_2 に連結演算子 $\cdot$ を施した式

   $(e_1)\cdot(e_2)$

   は，正規式である．正規式 $(e_1)\cdot(e_2)$ は，次の集合を表す．

   $$\left\{ w \,\middle|\, \begin{array}{l} x \text{ は正規式 } e_1 \text{ の表す集合の要素，} y \text{ は正規式 } e_2 \text{ の表す} \\ \text{集合の要素，} w = xy \end{array} \right\}.$$

6. 正規式 e に繰り返し演算子 $*$ を施した式

   $(e)*$

   は，正規式である．正規式 $(e)*$ は，次の集合を表す．

   $$\left\{ w \,\middle|\, \begin{array}{l} m \text{ は 0 以上の整数，} x_1, x_2, \cdots, x_m \text{ は正規式 } e \text{ の表す} \\ \text{集合の } m \text{ 個の要素，} w = x_1 x_2 \cdots x_m \text{．ただし } m = 0 \text{ の} \\ \text{場合，どんな正規式 } e \text{ に関しても，} w = \varepsilon \text{ とする} \end{array} \right\}.$$

7. 正規式 e に括弧を施した式

   (e)

   は，正規式である．正規式 (e) の表す集合は，正規式 e の表す集合と同一である．

8. 上記 1.～7. の規則から得られる式だけが，正規式である．
9. 演算子の優先順位の強さは，強い順に演算子 “*”，“・”， “+” である．また同一優先順位の演算子は，左から右の順序で評価する．
10. 規則 9. により誤解が起こらないときは，括弧 “(“と “)” を省略して得られる式も，正規式である． □

正規式 e が表す記号列集合を $\mathrm{L}(e)$ で表すことにする．正規式 e と f が $\mathrm{L}(e)=\mathrm{L}(f)$ のとき，$e=f$ と書くことにする．

例 2.1 (正規式の表す言語)

正規式 (a+b*·c)* は，正規式 ((a)+(((b)*) · (c)))* と同じ記号列の集合を表している．この式が表している記号列の集合を，定義に忠実に求めると以下のようになる．まず，正規式 a，b，c はそれぞれ記号列の集合

$\{a\}, \{b\}, \{c\}$

を表している．次に，正規式 b∗ は，記号列の集合

$\{\varepsilon, b, bb, bbb, \cdots\}$

を表す．そして，正規式 b∗·c は記号列の集合

$\{c, bc, bbc, bbbc, \cdots\}$

を表す．正規式 a+b∗·c は記号列の集合

$\{a, c, bc, bbc, bbbc, \cdots\}$

を表す．最終的に，正規式 (a+b∗·c)∗ は記号列の集合

$\{\varepsilon, a, c, aa, ac, bc, ca, \cdots\}$

を表していることになる．□

2.2　文脈自由文法

人間が無限個の文を生成できる理由を説明するために考え出されたモデルが句構造文法である．句構造文法の制限されたクラスである文脈自由文法は，言語の構文の記述法として広く使われている．

定義 2.3 (句構造文法)

句構造文法 (phrase structure grammar) G = (N, T, P, S) とは，次の 4 つ組である．(終端記号 T の上の句構造文法と呼ぶ.)

1. 書換えを行う対象となる，**非終端記号** (nonterminal symbol) と呼ばれる記号の有限集合 N.
2. 書換えを行うことができない，**終端記号** (terminal symbol) と呼ばれる記号の有限集合 T(集合 N と集合 T は共通要素はない)．終端記号と非終端記号を合わせて，**文法記号** (grammar symbol) と呼ぶ．

3. **生成規則** (production rule) の有限集合 P．生成規則は次の形をした書換え規則である．

   左辺 → 右辺

   ここで左辺は，少なくとも 1 個の非終端記号を含む，非終端記号と終端記号の長さ 1 以上の記号列，右辺は長さ 0 以上の非終端記号と終端記号の記号列である．左辺を右辺に書き換えることを示す規則の集合である．右辺の長さが 0 の生成規則を**空規則** (epsilon production) と呼ぶ．
4. **開始記号** (start symbol) と呼ぶ非終端記号 S．開始記号は書換えを開始する最初の非終端記号である．　□

定義 2.4 (文脈自由文法)
句構造文法の書換え規則において，左辺が必ず非終端記号 1 個となっているものを，**文脈自由文法** (context-free grammar) と呼ぶ．文脈自由文法の右辺は，長さ 0 以上の文法記号の記号列である．　□

非終端記号は通常，文法概念などを表し，終端記号は字句記号などを表す．終端記号のみからなる長さ 0 以上の記号列を，**文** (sentence) と呼ぶ．特に断わらない限り，文法の開始記号は生成規則の列の最初の生成規則の左辺の非終端記号とする．例えば，以下のような書換え規則を，人間が頭の中に獲得すると，さまざまな文が生成できると考えられる．

文	→	名詞句 動詞句			
名詞句	→	冠詞 名詞句	名詞	→	John
名詞句	→	名詞	名詞	→	Mary
動詞句	→	動詞	動詞	→	loves
動詞句	→	動詞 名詞句	···		

1 つの開始記号 (この場合は文という記号) から，有限個の書換え規則を用いて，さまざまな書換えを施すことにより，無限個または有限個の文が生成できることになる．

本書では，特に断わらない限り，以下次のような記法上の約束を用いることにする．

1. 非終端記号を，英大文字 A，B，C などで表す．

2. 終端記号を，英小文字 a，b，c や英字以外の特殊記号で表す．
3. 文法記号を，英大文字 X，Y，Z などで表す．
4. 文法記号の長さ 0 以上の列を，α，β，γ などで表す．
5. 終端記号の長さ 0 以上の列を，x，y，z などで表す．

例 2.2 (文脈自由文法)
次の例は文脈自由文法 G_1 である．

$$
\begin{aligned}
&G_1 = (N, T, P, S)\\
&N = \{S, A, B\}\\
&T = \{a, b\}\\
&P = \{\ 1)\ S \to aB,\\
&\qquad\ 2)\ S \to bA,\\
&\qquad\ 3)\ A \to a,\\
&\qquad\ 4)\ A \to aS,\\
&\qquad\ 5)\ A \to bAA,\\
&\qquad\ 6)\ B \to b,\\
&\qquad\ 7)\ B \to bS,\\
&\qquad\ 8)\ B \to aBB\}
\end{aligned}
$$

□

句構造文法における書換えは，開始記号 S のみからなる記号列から始め，次々と書換え規則に従い，記号列中の左辺に一致する 1 箇所の記号列の出現場所を，その記号を左辺に持つ書換え規則のいずれかの右辺に，書き換えることによって行われる．

定義 2.5 (導出)
句構造文法 G に対し，文法記号の有限列の集合上の関係 $\underset{G}{\Longrightarrow}$ を次のように定義する．

$$\gamma\alpha\delta \underset{G}{\Longrightarrow} \gamma\beta\delta \quad \overset{\text{定義}}{\Longleftrightarrow} \quad \alpha \to \beta \text{ は G の生成規則である．}$$

□

関係 $\underset{G}{\Longrightarrow}$ は，句構造文法 G の生成規則による 1 回の書換えを表す．0 回以上の書換えを，記法 $\overset{*}{\underset{G}{\Longrightarrow}}$ で表す．どの文法によるのか混乱が起こらないときは，G を省略して表す．開始記号 S のみからなる記号列から始め，書換えを有限回行い，終端記号列 w が得られたとき，文法 G に従って，1 つの文 w の**導出** (derivation) が行われたという．

定義 2.6 (生成言語)
句構造文法 G によって導出されるすべての文の集合を L(G) で表し，L(G) を，文法 G が生成する**言語** (language) と呼ぶ．記法では

$$\mathrm{L(G)} = \left\{ \mathrm{w} \;\middle|\; \mathrm{S} \overset{*}{\underset{G}{\Longrightarrow}} \mathrm{w},\ \mathrm{w}\ \text{は終端記号列} \right\}$$

である． □

終端記号列の集合 $\mathrm{L_0}$ が，少なくとも，1 つの文脈自由文法 G によって生成できるとき，$\mathrm{L_0}$は，**文脈自由言語** (context-free language) であるという．

例 2.3 (文脈自由言語)
文脈自由文法 $\mathrm{G_1}$について考える．文法 $\mathrm{G_1}$からは，例えば，以下に示すような 1 つの文 abbbaa が導出できる．この導出を導出 1 と呼ぶことにする．

$$\begin{array}{lll}
\mathrm{S} & \Rightarrow & \mathrm{aB} \\
 & \Rightarrow & \mathrm{abS} \\
 & \Rightarrow & \mathrm{abbA} \\
 & \Rightarrow & \mathrm{abbbAA} \\
 & \Rightarrow & \mathrm{abbbaA} \\
 & \Rightarrow & \mathrm{abbbaa}
\end{array}$$

文法 $\mathrm{G_1}$の場合，生成する言語は次のようになる．

$$\mathrm{L(G_1)} = \left\{ \mathrm{x} \;\middle|\; \begin{array}{l} \text{含まれる記号 a と b の個数がちょうど等しく，記号列} \\ \text{全体の長さが 2 以上で，記号 a と b からなる記号列 x} \end{array} \right\}.$$

したがって，文 abab や aabbab は，文法 $\mathrm{G_1}$の導出する文であるが，文 aaab や bbaaaa は導出することができない．

文を導出する書換えの過程は，1つの文に対して，一般には一通りとは限らない．例えば文 abbbaa に対して，文法 G_1の下で次のような導出も存在する．この導出を導出 2 と呼ぶことにする．

$$
\begin{array}{lll}
S & \Rightarrow & aB \\
 & \Rightarrow & abS \\
 & \Rightarrow & abbA \\
 & \Rightarrow & abbbAA \\
 & \Rightarrow & abbbAa \\
 & \Rightarrow & abbbaa
\end{array}
$$

□

文脈自由文法において，開始記号 S から 0 回以上の書換えで得られる文法記号の列を，**文形式** (sentential form) と呼ぶ．例えば，導出が実際に行われた文は，終端記号のみからなる文形式で，開始記号は非終端記号 1 つからなる文形式である．

文脈自由文法における書換えの過程において，毎回，文形式中の最も右側に位置する，非終端記号を書き換える導出を，**最右導出** (rightmost derivation) と呼ぶ．最右導出の過程で得られる文形式を，**右文形式** (right sentential form) と呼ぶ．

同様に，毎回，文形式中の最も左側に位置する非終端記号を書き換える導出を，**最左導出** (leftmost derivation) と呼ぶ．最左導出の過程で得られる文形式を，**左文形式** (left sentential form) と呼ぶ．例 2.3 の導出 1 は最左導出であり，導出 2 は最右導出である．

文脈自由文法の導出の過程で，絶対的な適用順序を問題にせず，適用位置，書換え規則の番号のみを問題にしたい場合は，図 2.1のような図を用いると便利である．

このような図を，**導出木** (derivation tree) または**構文解析木** (parse tree) と呼ぶ．導出木の根は開始記号で表し，書換え規則 $X_0 \rightarrow X_1\ X_2 \cdots X_n$ $(n \geq 0)$ を，図 2.2 のような木で表し，これらの木をつなぎ合わせることによって，導出木が得られる．文法 G_1 において，文 abbbaa は，上記のような導出木をただ 1 通り持つ．文法 G において，言語 L(G) に属すど

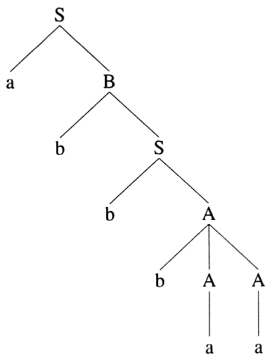


図 **2.1**　導出木

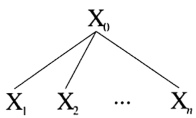


図 **2.2**　書換え規則

のような文も，導出木をただ 1 通り持つとき，文法 G をあいまいでない (unambiguous) 文法と呼ぶ．言語 L(G) に属するある文が，導出木を 2 通り以上持つとき，文法 G をあいまいな (ambiguous) 文法という．文法 G_1 は，例えば，文 bbaaba に対して，図 2.3 のように 2 通りの導出木を持つため，あいまいな文法である．

最左導出における適用生成規則の番号の列を，**左解析列** (left parse) と呼ぶ．最右導出における適用生成規則の番号の列の左右反転列を，**右解析列** (right parse) と呼ぶ．例えば，文法 G_1の下で，文 abbbaa の 1 つの左解析列は，172533 であり，1 つの右解析列は，335271 である．あいまいでない文脈自由文法は，任意の文に対し，高々1 つの最左導出と高々1 つの最右導出を持つため，左解析列も右解析列も高々1 つ存在する．

例 2.4 (文脈自由文法)
基本的な言語を生成する文法をいくつか示す．いずれも無限個の文を持った言語を生成している．

1. 言語 $\{(\mathrm{ab})^n \mid n \geq 1\}$ を生成する以下の文法では，

   1) S → S a b

   2) S → a b

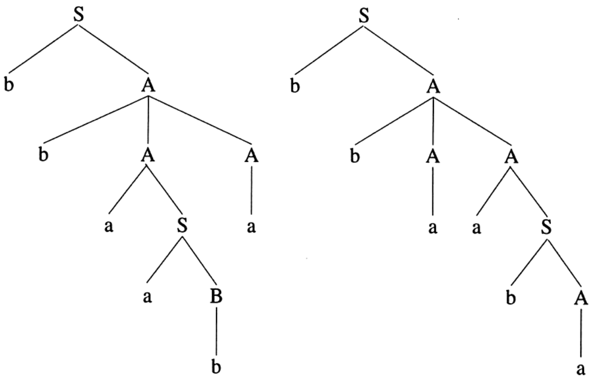


図 **2.3**　2 通りの導出木

S 自身が右辺の最左位置に出現する**左再帰** (left-recursive) 型の非終端記号となっている．

同じ言語を生成する以下の文法では，

1) S → a b S

2) S → a b

S 自身が右辺の最右位置に出現する**右再帰** (right-recursive) 型の非終端記号となっている．

2. 言語 $\{\mathrm{a}^m\mathrm{b}^n \mid m \geq 1,\ n \geq 1\}$ を生成する以下の文法では，

1) S → A B

2) A → A a

3) A → a

4) B → B b

5) B → b

m と n の等しいものや等しくないものを導出することができる．

3. 言語 $\{\mathrm{a}^n\mathrm{b}^n \mid n \geq 1\}$ を生成する以下の文法では，

1) S → a S b

2) S → a b

S 自身が右辺の途中位置に出現する自己埋め込み型の非終端記号となっている. □

2.3 句構造文法の実例

文脈自由文法の持つ性質の一般化や特殊化を行った句構造文法のいくつかのクラスについて説明する.

1. 文脈依存文法
   書換え規則 "左辺 → 右辺" において，右辺の長さが，必ず左辺の長さ以上となっているものを，**文脈依存文法** (context-sensitive grammar) と呼ぶ．特に，右辺の長さは，必ず 1 以上である．文法記号の出現する文脈に応じて書換えができるので，この名で呼ばれる．空規則を持たない文脈自由文法は文脈依存文法である.

3. 正規文法
   書換え規則 "左辺 → 右辺" において，左辺が，必ず非終端記号 1 個となっており，右辺は，長さ 0 以上の終端記号の記号列に，0 個か 1 個の非終端記号をつなげた形式となっているものを，**正規文法** (regular grammar) と呼ぶ．正規文法は文脈自由文法である.

空規則を 1 個以上持つ文脈自由文法は，生成する言語を変化させずに高々 1 個の空規則のみを持つ文脈自由文法に書き直すことができる．しかも，この空規則の左辺は，左辺にのみ出現する開始記号とすることができる.

また，その生成言語に空列を含まない文脈自由文法や正規文法は生成する言語を変化させずに次のような標準形に書き直すことができる.

1. 文脈自由文法の Chomsky **の標準形**
   生成規則の右辺は非終端記号 2 個または終端記号 1 個のいずれかである.
2. 文脈自由文法の Greibach **の標準形**
   生成規則の右辺は，終端記号 1 個に 0 個以上の非終端記号の列をつなげたものである.
3. **正規文法の標準形**

生成規則の右辺は終端記号 1 個に 0 個か 1 個の非終端記号をつなげたものである．

一般の句構造文法を，0 型文法 (type 0 grammar) と呼ぶ．以下，0 型文法，文脈依存文法，文脈自由文法，正規文法の順にいくつかの例を示す．

例 2.5 (0 型文法)

0 型文法を用いると，言語 { wcw | w は a と b の長さ 0 以上の列 } を生成する文法が，以下のように記述できる．

S→ATB	RaLa→LaRa	ALa→aA
T→TU	RaLb→LbRa	ALb→bA
T→ ε	RbLa→LaRb	RaB→Ba
U→LaRa	RbLb→LbRb	RbB→Bb
U→LbRb	AB→c	

例えば，abcab を導出する過程は，次のようになる．

S	⇒	ATB	⇒	ATUB
	⇒	ATUUB	⇒	AUUB
	⇒	ALaRaUB	⇒	ALaRaLbRbB
	⇒	ALaLbRaRbB	⇒	aALbRaRbB
	⇒	abARaRbB	⇒	abARaBb
	⇒	abABab	⇒	abcab

□

例 2.6 (0 型文法)

言語 $\{a^{2^n} \mid n \geq 1\}$ を生成する文法も 0 型文法を用いると，次のように記述できる．

S → LCaR	aD → Da
Ca → aaC	LD → LC
CR → DR	aE → Ea
CR → E	LE → ε

例えば，aaaa を導出する過程は以下のようになる．

$$
\begin{aligned}
\mathrm{S} &\Rightarrow \mathrm{LCaR} &&\Rightarrow \mathrm{LaaCR}\\
&\Rightarrow \mathrm{LaaDR} &&\Rightarrow \mathrm{LaDaR}\\
&\Rightarrow \mathrm{LDaaR} &&\Rightarrow \mathrm{LCaaR}\\
&\Rightarrow \mathrm{LaaCaR} &&\Rightarrow \mathrm{LaaaaCR}\\
&\Rightarrow \mathrm{LaaaaE} &&\Rightarrow \mathrm{LaaaEa}\\
&\Rightarrow \mathrm{LaaEaa} &&\Rightarrow \mathrm{LaEaaa}\\
&\Rightarrow \mathrm{LEaaaa} &&\Rightarrow \mathrm{aaaa}
\end{aligned}
$$

□

例 2.7 (文脈依存文法)
文脈依存文法を用いると，言語 $\{\mathrm{a}^n\mathrm{b}^n\mathrm{c}^n \mid n \geq 1\}$ を生成する文法が次のように記述できる．

$$
\begin{aligned}
&\mathrm{S} \rightarrow \mathrm{aSBC} && \mathrm{bB} \rightarrow \mathrm{bb}\\
&\mathrm{S} \rightarrow \mathrm{aBC} && \mathrm{bC} \rightarrow \mathrm{bc}\\
&\mathrm{aB} \rightarrow \mathrm{ab} && \mathrm{cC} \rightarrow \mathrm{cc}\\
&\mathrm{CB} \rightarrow \mathrm{BC} &&
\end{aligned}
$$

例えば，aaabbbccc を導出する過程は次のようになる．

$$
\begin{aligned}
\mathrm{S} &\Rightarrow \mathrm{aSBC} &&\Rightarrow \mathrm{aaSBCBC}\\
&\Rightarrow \mathrm{aaaBCBCBC} &&\Rightarrow \mathrm{aaaBBCCBC}\\
&\Rightarrow \mathrm{aaaBBCBCC} &&\Rightarrow \mathrm{aaaBBBCCC}\\
&\Rightarrow \mathrm{aaabBBCCC} &&\Rightarrow \mathrm{aaabbBCCC}\\
&\Rightarrow \mathrm{aaabbbCCC} &&\Rightarrow \mathrm{aaabbbcCC}\\
&\Rightarrow \mathrm{aaabbbccC} &&\Rightarrow \mathrm{aaabbbccc}
\end{aligned}
$$

□

例 2.8 (算術式の文法)
中置記法，前置記法，後置記法の算術式を定義する文脈自由文法の例をいくつか示す．終端記号 i は，変数あるいは定数を示している．

1. 中置記法の算術式を定義する文法の例

1) $S \to S + S$	4) $S \to S \,/\, S$
2) $S \to S - S$	5) $S \to (\, S \,)$
3) $S \to S \times S$	6) $S \to i$

2. 中置記法の算術式を定義するもう 1 つの文法の例

1) $E \to E + T$	5) $T \to T \,/\, F$
2) $E \to E - T$	6) $T \to F$
3) $E \to T$	7) $F \to (\, E \,)$
4) $T \to T \times F$	8) $F \to i$

3. 中置記法の算術式を定義するさらにもう 1 つの文法の例

1) $E \to T\ E'$	6) $T' \to \times\ F\ T'$
2) $E' \to +\ T\ E'$	7) $T' \to /\ F\ T'$
3) $E' \to -\ T\ E'$	8) $T' \to \varepsilon$
4) $E' \to \varepsilon$	9) $F \to (\, E \,)$
5) $T \to F\ T'$	10) $F \to i$

4. 前置記法の算術式を定義する文法の例

1) $S \to +\ S\ S$	4) $S \to /\ S\ S$
2) $S \to -\ S\ S$	5) $S \to i$
3) $S \to \times\ S\ S$	

5. 後置記法の算術式を定義する文法の例

1) $S \to S\ S\ +$	4) $S \to S\ S\ /$
2) $S \to S\ S\ -$	5) $S \to i$
3) $S \to S\ S\ \times$	

□

例 2.9 (正規文法)
言語 $\{a^m b^n \mid m \geq 1,\ n \geq 1\}$ を生成する正規文法は以下のようになる．

1) S → a A	4) A → b B
2) A → b	5) B → b
3) A → a A	6) B → b B

□

2.4　文脈自由文法の上の計算

文脈自由文法の生成規則の形についてまず述べ，次に文脈自由文法の上のいくつかの関数の計算法を説明する．これらの関数は，以後の章で必要となる関数である．

文脈自由文法の生成規則の形について用語を定義しよう．文脈自由文法によっては，不用な非終端記号や終端記号を含む場合がある．不用な記号とは，開始記号からその記号へ0回以上の書換えで到達することができない，あるいはその記号から0回以上の書換えで終端記号列を得ることができない記号である．

定義 2.7 (開始記号から到達可能な文法記号)
文脈自由文法 G において，開始記号から**到達可能な文法記号**とは，次のような記号である．

1. 開始記号 S は，開始記号から到達可能な文法記号である．
2. 記号 X_0 が開始記号から到達可能な文法記号で，生成規則 $X_0 \rightarrow X_1\ X_2 \cdots X_n$ が存在するなら，文法記号 X_1，X_2，$\cdots$，X_n は開始記号から到達可能な文法記号である．□

定義 2.8 (終端記号から到達可能な文法記号)
文脈自由文法 G において，終端記号から**到達可能な文法記号**とは，次のような記号である．

1. 終端記号 X は，終端記号から到達可能な文法記号である．
2. 文法記号 X_1，X_2，$\cdots$，X_n が終端記号から到達可能な文法記号で，生成規則 $X_0 \rightarrow X_1\ X_2 \cdots X_n$ が存在するなら，記号 X_0 は終端記号から到達可能な文法記号である．□

定義 2.9 (不用な文法記号)

文脈自由文法 G において，開始記号から到達可能でない，あるいは終端記号から到達可能でない文法記号を，**不用な** (useless) **文法記号**という．これらの記号は通常，終端記号の集合や非終端記号の集合から除き，不用な記号を含む生成規則も生成規則の集合から取り除いて考えることにする．　□

一般には文法 G の開始記号 S の出現場所は，1 番目の生成規則の左辺のみならず，自由に出現してよい．開始記号の出現場所を明確に制限したい場合に，もとの文法 G に対して，新しい非終端記号 S′ と，新しい 0 番目の生成規則，

$$\mathrm{S}' \to \mathrm{S}$$

を導入して，S′ を新しい開始記号とする文法 G′ を用いることがある．この文法 G′ を，文法 G の**拡大文法** (augmented grammar) という．左辺が非終端記号 A である生成規則を，A **規則** (A-production) という．

文形式などからどのような終端記号列が導出可能かを調べるために必要な関数が以下の FIRST_k である．

定義 2.10 ($\mathbf{FIRST}_k$)

1 つの文脈自由文法 G のもとで，非負整数 k に対し，与えられた G の文法記号の列 α に対して，**FIRST** $[\mathbf{G}]_k(\boldsymbol{\alpha})$ を次のように定義する．

$$\mathrm{FIRST}[\mathrm{G}]_k(\alpha) = \left\{ \mathrm{y} \;\middle|\; \begin{array}{l} \alpha \text{ からの 0 回以上の導出で得られる終端記} \\ \text{号列が x である．x の長さが } k \text{ 以上なら，y} \\ \text{は x の先頭 } k \text{ 記号の列．x の長さが } k \text{ 未満} \\ \text{ならば，y は x 自身である} \end{array} \right\}$$

混乱が起こらない場合は，$\mathrm{FIRST}[\mathrm{G}]_k(\alpha)$ を $\mathrm{FIRST}_k(\alpha)$ と表すことにする．
□

$\mathrm{FIRST}_1(\alpha)$ の計算法を説明する．G に空規則がなく，α の長さが 1 の場合からまず説明する．

方法 2.1 (空規則無しの $\mathbf{FIRST_1}$)

入力: 空規則を持たない文脈自由文法 G.

出力: G の各文法記号 X の $\mathrm{FIRST}_1(\mathrm{X})$.

手順:

1. それぞれの非終端記号 A について，終端記号の集合 First(A) を空集合とする.
2. それぞれの非終端記号 A について，規則 $\mathrm{A} \rightarrow \mathrm{a}\alpha$ なる形の規則が存在し，終端記号 a がまだ First(A) の要素でなければ，First(A) に終端記号 a を加える.
3. 次の (i) を，どの非終端記号の First についても，新しい要素がつけ加わらなくなるまで繰り返す. 終了したらステップ 4. へ行く.
   (i) それぞれの非終端記号 A について，規則 $\mathrm{A} \rightarrow \mathrm{B}\alpha$ なる形の規則が存在するならば，First(B) の要素のうちまだ First(A) に含まれていない終端記号を，First(A) に加える.
4. 各非終端記号 A について，$\mathrm{FIRST}_1(\mathrm{A}) = \mathrm{First}(\mathrm{A})$ とする. 各終端記号 a について，$\mathrm{FIRST}_1(\mathrm{a}) = \{\mathrm{a}\}$ とする. □

次に空規則が存在し，α の長さが 1 のときの $\mathrm{FIRST}_1(\alpha)$ の計算法を説明する.

定義 2.11 ($\boldsymbol{k}$ 記号生成 (N))

1 つの文脈自由文法 G のもとで，非負整数 k に対し，与えられた G の非終端記号 N について，$\boldsymbol{k}$ **記号生成 [G](N)** を次のように定義する.

$$k\ \text{記号生成}\ [\mathrm{G}](\mathrm{N}) = \left\{ \mathrm{x} \;\middle|\; \begin{array}{l} \text{N からの 1 回以上の導出で得られる} \\ \text{ちょうど長さ } k \text{ の終端記号列 x} \end{array} \right\}.$$

混乱が起こらないときは，k 記号生成 [G](N) を k 記号生成 (N) と表す. □

方法 2.2 (0 記号生成検査)

入力: 一般の文脈自由文法 G.

出力: G の各非終端記号 N の 0 記号生成 (N).

手順:

1. それぞれの非終端記号 A について，記号列の集合 Epsilon(A) を空集合とする．
2. それぞれの非終端記号 A について，$A \to \varepsilon$ なる形の規則が存在するならば，Epsilon(A) に ε を加える．
3. 次のステップ (i) を，どの非終端記号の Epsilon についても変更がなくなるまで繰り返す．
   (i) それぞれの非終端記号 A について，$A \to B_1\ B_2 \cdots B_n$ (n は 1 以上) なる形の規則が存在し，$\mathrm{Epsilon}(B_1) = \{\varepsilon\}$，$\mathrm{Epsilon}(B_2) = \{\varepsilon\}$，$\cdots$，$\mathrm{Epsilon}(B_n) = \{\varepsilon\}$ で，Epsilon(A) が空集合ならば，Epsilon(A) $= \{\varepsilon\}$ とする．
4. 各非終端記号 N の 0 記号生成 (N) を Epsilon(A) とする．　□

方法 2.3 (空規則有りの $\mathbf{FIRST_1}$)

入力: 一般の文脈自由文法 G.

出力: G の各文法記号 X の $\mathrm{FIRST}_1(X)$.

手順:

1. それぞれの文法記号 X について，G の生成規則の集合から空規則を消去して得られる文法 G′ に関する $\mathrm{FIRST}[G']_1(X)$ を First(A) の初期値とする．また 0 記号生成 (A) $= \{\varepsilon\}$ ならば，さらに，FIRST(A) に ε を加える．
2. 次の (i) および (ii) を，どの非終端記号の First についても新しい終端記号が加わらなくなるまで繰り返す．
   (i) それぞれの非終端記号 A について，$A \to B_1\ B_2 \cdots B_i\ C\ \alpha$ という形の規則が存在し，しかも 0 記号生成 $(B_1) = \{\varepsilon\}$，0 記号生成 $(B_2) = \{\varepsilon\}$，$\cdots$，0 記号生成 $(B_i) = \{\varepsilon\}$ ならば，First(C) の要素のうちまだ First(A) に含まれていない終端記号を，First(A) に加える．
   (ii) それぞれの非終端記号 A について，$A \to B_1\ B_2 \cdots B_j\ a\ \alpha$ という形の規則が存在し，しかも 0 記号生成 $(B_1) = \{\varepsilon\}$，0 記号生成 $(B_2) = \{\varepsilon\}$，$\cdots$，0 記号生成 $(B_j) = \{\varepsilon\}$ ならば，終端記号 a がまだ Frist(A) に含まれていない場合，その終端記号 a を，First(A) に加える．　□

定義 2.12 (限定的結合演算子)

$L_1 \oplus_k L_2 = \{\ \mathrm{FIRST}_k(\mathrm{xy}) \mid$ x は L_1 の要素，y は L_2 の要素 $\}$.　□

方法 2.4 (文法記号列の FIRST$_1$)

入力: 文法記号列 $\mathrm{X}_1\mathrm{X}_2\cdots\mathrm{X}_n$.

出力: $\mathrm{FIRST}_1(\mathrm{X}_1\mathrm{X}_2\cdots\mathrm{X}_n)$.

手順:

$$\mathrm{FIRST}_1(\mathrm{X}_1\mathrm{X}_2\cdots\mathrm{X}_n) = \mathrm{FIRST}(\mathrm{X}_1) \oplus_1 \mathrm{FIRST}(\mathrm{X}_2) \oplus_1 \cdots \oplus_1 \mathrm{FIRST}(\mathrm{X}_n).$$

□

定義 2.13 (右端記号)

記号 \$ は G の終端記号に存在しない新しい特別な終端記号で，入力列の右端を示す記号として用いる．記号 \$ を**右端記号**と呼ぶ．　□

定義 2.14 (FOLLOW$_k$)

1 つの文脈自由文法 G のもとで，非負整数 k に対し，与えられた G の非終端記号 N に対して，**FOLLOW[G]$_k$(N)** を次のように定義する．

$$\mathrm{FOLLOW[G]}_k(\mathrm{N}) = \left\{ \mathrm{x} \;\middle|\; \begin{array}{l} \text{導出 } \mathrm{S\$}^k \overset{*}{\Longrightarrow} \alpha\mathrm{N}\beta \overset{*}{\Longrightarrow} \mathrm{w} \text{ が存在し,} \\ \mathrm{x} \text{ は } \mathrm{FIRST[G]}_k(\beta) \end{array} \right\}.$$

混乱が起こらないときは，$\mathrm{FOLLOW[G]}_k(\mathrm{N})$ を $\mathrm{FOLLOW}_k(\mathrm{N})$ と表す．□

方法 2.5 (FOLLOW$_1$の計算法)

入力: 一般の文脈自由文法 G.

出力: G の各非終端記号 N の $\mathrm{FOLLOW}_1(\mathrm{N})$.

手順:

1. それぞれの非終端記号 A について，終端記号列の集合 Follow(A) を空集合とする．
2. 開始記号 S に対し，Follow(S) に \$ を加える．
3. 次の (i) を，どの非終端記号の Follow についても新しい要素がつけ加わらなくなるまで繰り返す．
   (i) それぞれの規則 $\mathrm{A} \to \alpha\mathrm{B}\beta$ と Follow(A) のそれぞれの要素 x に対し，
   $$\mathrm{FIRST}_1(\beta) \oplus_1 \{\mathrm{x}\}$$

の要素のうち，まだ Follow(B) に含まれていない終端記号を，Follow(B) に加える．

4. 各非終端記号 N の $\mathrm{FOLLOW}_1(\mathrm{N})$ を Follow(N) とする．　□

2.5　非決定性表現

プログラムの次の動作が一意的でなく，同一状況で 2 個以上の動作のうちから 1 つを選ぶことが許されるプログラム表現を，**非決定性** (nondeterministic) **プログラム**と呼ぶ．これに対し，プログラムの次の動作が高々1 通りであるプログラム表現を，**決定性** (deterministic) **プログラム**と呼ぶ．

非決定性プログラムは通常のプログラム A に並列選択文を追加したものと考えられる．プログラム A の実行中，n 個の動作から 1 つの動作を選択する次のような並列選択文に到達すると，

並列選択 (動作 1 | ⋯ | 動作 n)

同一プログラム A と入力データが n 個コピーされ，1 番目のプログラムは動作 1 を実行し，⋯，n 番目のプログラムは動作 n を実行し，そして n 個のプログラムがそれぞれ並列選択文以後の実行の続きを行う．

プログラムの各コピーは，成功条件か失敗条件に到達すると，終了する．非決定性プログラム全体は，少なくとも 1 つのコピーが成功条件に到達できる場合，**成功**したといい，どのコピーも成功条件に到達できない場合，**失敗**したという．

非決定性プログラムを用いると，探索問題などの解を簡潔に表現できる．例えば，迷路の出口探索は次のような非決定性プログラムで表現できる．

---

1. 入口に立つ．
2. 以下を永久に繰り返す．
   - 2-1. 通路に沿って 1 単位進み，地点 P に到達する．
   - 2-2. P が行き止まりまたは入口ならば失敗．
   - 2-3. P が出口ならば成功．
   - 2-4. P が分岐点ならば，並列選択 (左折する | 直進する | 右折する)．

非決定性プログラムは，決定性プログラムと異なり，1通りの可能性を作り出し，成功か失敗かを検査すればよいので，記述は極めて簡潔となる．非決定性プログラムの実行は，通常はバックトラックを用いる決定性プログラムに変換して行うことが多い．効率を無視することが許されるなら，非決定性プログラムは，プロセスのフォーク命令を用いて，計算機システム上で極めて忠実にその実行を行うことができる．

第3章　字句解析の方法

文字の列を字句記号の列に区切るプログラムを，字句解析プログラムという．字句解析プログラムの構成には，与えられた文字列が指定されたパターンを満たすかどうかを判定する字句認識プログラムが必要となる．本章では，字句認識系の2つの構成法を述べ，最後に字句解析系の構成法を述べる．

3.1　字句認識系

正規式で指定されたパターンを，与えられた記号列が満たすかどうかを認識するプログラムを構成する問題を検討する．例えば，正規式 $a{\cdot}a{*}{\cdot}b{\cdot}b{*}$ が表すパターンを満たす記号列を認識するプログラムを手書きで作成する場合を考えると，プログラム 3.1 のようになる．

プログラム 3.1

```
w : 記号列変数
w ← 入力列
goto 0
0:
if 空 (w) then 出力 (“no”); 停止 () fi
case 先頭 (w) in
  ’a’: w ← 残り (w)
       goto 1
```

```
   'b': w ← 残り (w)
        goto 3
esac
1:
if 空 (w) then 出力 ("no"); 停止 () fi
case 先頭 (w) in
   'a': w ← 残り (w)
        goto 1
   'b': w ← 残り (w)
        goto 2
esac
2:
if 空 (w) then 出力 ("yes"); 停止 () fi
case 先頭 (w) in
   'b': w ← 残り (w)
        goto 2
   'a': w ← 残り (w)
        goto 3
esac
3:
if 空 (w) then 出力 ("no"); 停止 () fi
case 先頭 (w) in
   'a': w ← 残り (w)
        goto 3
   'b': w ← 残り (w)
        goto 3
esac
```

このプログラムは，入力列の先頭1記号の判定 (先頭 (w))，入力列全体の空列判定 (空 (w))，入力列の先頭1記号の消費 (w ← 残り (w))，有限個のラベルへのジャンプ，出力，停止などから構成されている．このようなプログラムは有限個の状態を示すラベルで区別してプログラムの制御が進行するので，**有限状態プログラム**と呼ぶ．特に，この場合は，状態0から状態3までの4状態からなる有限状態プログラムである．状態0を**開始状態**，"yes"を出力する状態を**終了状態**という．

有限状態プログラムを１つ１つの正規式ごとに作成するのは非能率である．そこで共通部分を駆動プログラムとして用意して，正規式ごとに変わる部分を表の形で保持することを試みる．このようなプログラムの作り方を，**表駆動型プログラム**という．

プログラム 3.1 に対応する表駆動型プログラムは，駆動プログラム 3.2 と表 3.1 となる．プログラム 3.2 の中で，遷移 $(q,$ 先頭 (w)) は表 3.1 中の状態が q ，入力が先頭 (w) の行の行き先状態を返す関数である．表 3.1 のように状態と入力に対し行き先状態を定義した表を，**状態遷移表**と呼ぶ．

プログラム 3.2

```
q : 状態変数
w : 記号列変数
q ← 0
w ← 入力列
while not  (空 (w))  do
  q ← 遷移 (q, 先頭 (w))
  if  未定義 (q)  then  停止 ()  fi
  w ← 残り (w)
od
if  終了 (q)
then  出力 (“yes”); 停止 ()
else  出力 (“no”); 停止 ()
fi
```

例として，入力列 aaabbb の認識の様子を表 3.2 に示す．開始状態 0 で，入力記号 a を消費して，状態 1 に移る．状態 1 で，入力記号 a を消費して，再び状態 1 に移る．状態 1 で，入力記号 a を消費して，再び状態 1 に移る．状態 1 で，入力記号 b を消費して，状態 2 に移る．状態 2 で，入力記号 b を消費して，再び状態 2 に移る．状態 2 で，入力記号 b を消費して，再び状態 2 に移る．入力列をすべて消費したときの状態が，終了状態とマークされているので，正しく入力を受け付けたことになる．

表 3.1　状態遷移表

状態	入力	行き先状態
0	a	1
0	b	3
1	a	1
1	b	2(終了)
2	b	2(終了)
2	a	3
3	a	3
3	b	3

表 3.2　認識過程

入力列	状態
aaabbb	0
aabbb	1
abbb	1
bbb	1
bb	2
b	2
ε	2(終了)

複雑な正規式の場合，直接に有限状態プログラムを書くことが難しい場合がある．すなわち，

1. 同一記号を消費してある状態へ遷移させたいし，別の状態へも遷移させたい，あるいは
2. 入力記号を全く消費しないである状態へ遷移させたいし，記号を消費して別の状態へも遷移させたい

といったプログラムを書きたい場合が起こってくる．入力記号を全く消費しないで別の状態へ遷移することを，**ε 遷移**という．関数 “遷移” の第 2 引数が記号および ε で引けるようにし，かつ関数 “遷移” の返す値を状態番号の集合と改良すると，新しい駆動プログラムはプログラム 3.3 のようになる．

プログラム 3.3

```
q : 状態変数
w : 記号列変数
q ← 0
w ← 入力列
while true do
  {r1, ···, rm} ← 遷移 (q, ε)
  if m = 0 then goto はじまり fi
  並列選択 (goto はじまり | q ← r1; goto おわり | ··· | q ← rm; goto おわり
  はじまり:
  if (空 (w)) then goto さいご fi
```

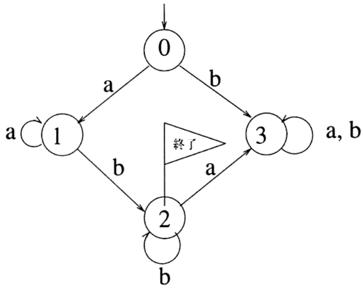


図 3.1　状態遷移図

　$\{q_1, \cdots, q_n\} \leftarrow$ 遷移 (q, 先頭 (w))
　if $n = 0$ **then** 停止 () **fi**
　並列選択 ($q \leftarrow q_1 \mid \cdots \mid q \leftarrow q_n$)
　w ← 残り (w)
　おわり:
od
さいご:
if 終了 (q)
then 出力 (“yes”); 停止 ()
else 出力 (“no”); 停止 ()
fi

一般的である駆動プログラム 3.3 を固定すると，正規式を認識する一般の有限状態プログラムを作成することは，状態遷移表を作ることと同一視できる．

状態遷移表は，状態の集合を節点の集合とするラベル付き有向グラフで実現できる．このラベル付き有向グラフを，**状態遷移図**という．表 3.1 に対応するラベル付き有向グラフは図 3.1 のようになる．

一般の有限状態プログラムを，**有限オートマトン**という．状態遷移表をもとに，正規式の認識に関係する 3 種類の有限オートマトンである，決定性有限オートマトン，非決定性有限オートマトン，ε 遷移を許す決定性有限オートマトンを説明する．これらは，一見能力の異なる有限オートマトンのよう

表 3.3　状態遷移表

状態	入力	状態
	$\cdots$	
i	X	j_1
i	X	j_2
	$\cdots$	
i	X	j_n
	$\cdots$	

表 3.4　状態遷移表

状態	入力	状態
	$\cdots$	
i	ε	k_1
i	ε	k_2
	$\cdots$	
i	ε	k_m
	$\cdots$	

に見えるが，後に示すように言語を認識する能力は一致する．

1つの有限オートマトンは，状態の有限集合と状態遷移の関係の表からなる．これらの状態のうち，番号0の状態を，**開始状態**と呼び，認識の最初の状態を表す．状態集合のうちの0個以上の状態は，**終了状態**と呼ばれる状態である．

遷移状態の遷移関係は，状態 i と入力記号 X に対して，その遷移する先の状態の集合を与える遷移関数 $\delta(i, \mathrm{X})$ によって定義される．遷移関数 $\delta(i, \mathrm{X}) = \{j_1, j_2, \cdots, j_n\}$ のとき，遷移関係の表に表 3.3 のように n 個の行を記入する．遷移先の状態の集合が空集合のとき，つまり $\delta(i, \mathrm{X}) = \phi$ のとき，行き先の状態は未定義と考える．

遷移先の状態の集合 $\delta(i, \mathrm{X})$ が，どのような状態 i と入力記号 X に対しても，高々 1 個の要素からなる集合のとき，この有限オートマトンを，**決定性** (deterministic) **オートマトン**であるという．

これに対し，遷移する先の状態の集合 $\delta(i, \mathrm{X})$ の個数が0個以上の一般の個数のとき，**非決定性** (nondeterministic) **オートマトン**であるという．

ε 遷移を許す非決定性オートマトンでは，さらに入力記号を見ないで遷移することを許す．すなわち，状態の遷移関係において，状態 i と入力記号 X に対して，その遷移する先の状態の集合を与える遷移関数 $\delta(i, \mathrm{X})$ のほかに，入力記号を消費しないで遷移するときの遷移関数 $\delta(i, \varepsilon) = \{k_1, k_2, \cdots, k_m\}$ が定義できる．遷移関係の表には表 3.4 のように ε 遷移の先を記入する．

有限オートマトンがどのような記号列の集合を認識するかを説明するために，これらの有限オートマトンの形式的定義を述べる．

定義 3.1 (非決定性有限オートマトン)
1つの**非決定性有限オートマトン** M = (Q, T, δ, i, F) は，次の5つの構成要素からなる．

1. 状態の有限集合 Q.
2. 入力記号の有限集合 T.
3. 状態の遷移関数 δ. 遷移関数 δ は，状態 q と入力記号 X の2つ組 (q, X) に対し，状態の集合 $\delta(q, \mathrm{X}) = \{ q_1, q_2, \cdots, q_n\}$ を対応付ける．状態 q と入力記号 X について，集合 $\delta(q, \mathrm{X})$ が空集合のとき，$\delta(q, \mathrm{X})$ は**未定義**であるという．
4. 開始状態と呼ばれる，Q の1つの要素 i.
5. 終了状態の集合と呼ばれる，Q の部分集合 F. F の要素は終了状態と呼ばれる． □

集合 A の上の関係 R を 0 回以上たどって得られる関係を記法 R$*$ で表すことにする．関係 R$*$ は関係 R の**反射的推移的閉包**と呼ばれる．

定義 3.2 (非決定性有限オートマトンの受理言語)

1. 状態 q と入力記号列 w の2つ組 (q, w) を，非決定性有限オートマトンの**計算状況**と呼ぶ．
2. 非決定性有限オートマトンの計算状況の上の遷移関係 $\vdash$ を，次のように定義する．
$$(q_1, \mathrm{w}_1) \vdash (q_2, \mathrm{w}_2) \overset{\text{定義}}{\Longleftrightarrow} \mathrm{w}_1 = \mathrm{X}{\cdot}\mathrm{w}_2,\ q_2 \text{ は } \delta(q_1, \mathrm{X}) \text{ の要素}.$$
3. 非決定性有限オートマトン M = (Q, T, δ, i, F) の**受理言語** L(M) は次のように定義する．
$$\mathrm{L(M)} = \{ \mathrm{w} \mid (i, \mathrm{w}) \vdash * (q, \varepsilon), q \text{ は F の要素} \}.$$ □

定義 3.3 (決定性有限オートマトン)
1つの非決定性有限オートマトン M = (Q, T, δ, i, F) の遷移関数 δ が，次の性質を満たすとき，**決定性有限オートマトン**であるという．

すべての状態 q と入力記号 X について，集合 $\delta(q, \mathrm{X})$ が 1 個か 0 個の要素からなる集合である．□

定義 3.4 (ε 遷移を許す非決定性有限オートマトン)

1. **ε 遷移を許す非決定性有限オートマトン** $\mathrm{M} = (\mathrm{Q}, \mathrm{T}, \delta, i, \mathrm{F})$ とは，非決定性有限オートマトンの遷移関数 δ を次のように拡張したものである．
   状態の遷移関数 δ は，状態 q と入力記号 X の組および状態 q と空列記号 ε の組に対し，状態の集合 $\delta(q, \mathrm{X}) = \{q_1, q_2, \cdots, q_n\}$ と状態の集合 $\delta(q, \varepsilon) = \{r_1, r_2, \cdots, r_m\}$ とを対応付ける．ある状態 q と入力記号 X について，集合 $\delta(q, \mathrm{X})$ が空集合のとき，状態 q の遷移先は，**未定義**であるという．
2. 状態 q と入力記号列 w の 2 つ組 (q, w) を，ε 遷移を許す非決定性有限オートマトンの**計算状況**と呼ぶ．
3. ε 遷移を許す非決定性有限オートマトンの計算状況の上の遷移関係 $\vdash$ は，次のように定義できる．
   $(q_1, \mathrm{w}_1) \vdash (q_2, \mathrm{w}_2) \overset{\text{定義}}{\Longleftrightarrow} \mathrm{w}_1 = \mathrm{X}\cdot\mathrm{w}_2$，$q_2$ は $\delta(q_1, \mathrm{X})$ の要素，
   X は空列か入力記号 1 個．
   特に，$\mathrm{X} = \varepsilon$ の遷移を，**ε 遷移** (ε-move) という．また y, w を入力記号の長さ 0 以上の列として
   $(q_1, \mathrm{y}\cdot\mathrm{w}) \vdash * (q_2, \mathrm{w})$
   のとき，状態 q_1 から (記号列 y を消費して) 状態 q_2 に**到達する**という．
4. ε 遷移を許す非決定性有限オートマトン $\mathrm{M} = (\mathrm{Q}, \mathrm{T}, \delta, i, \mathrm{F})$ の**受理言語** L(M) は，次のように定義できる．
   $\mathrm{L(M)} = \{\mathrm{w} \mid (i, \mathrm{w}) \vdash * (q, \varepsilon), q \text{ は F の要素}\}$．□

有限オートマトンの遷移関数が，全関数でないとき，すなわち，遷移先の未定義の場合が存在するとき，**不完全定義型**の有限オートマトンという．

不完全定義型の有限オートマトンが与えられたときには，新しい状態 q' を 1 つ付け加え，

1. それぞれの未定義の遷移先をすべて q' とし，
2. q' でのいかなる遷移先も q' とする

ことにより，**完全定義型**の有限オートマトンに直すことができる．

3.2　間接的構成法

正規式から有限オートマトンを作る間接的構成法では，与えられた正規式から，まず ε 遷移を許す非決定性有限オートマトンを作る．そしてこの ε 遷移を許す非決定性有限オートマトンから，完全定義型の決定性有限オートマトンを作り，さらに最小状態数の完全定義型決定性有限オートマトンを作る．

間接的構成法で正規式 a·a∗·b·b∗ を認識する有限オートマトンを構成してみよう．まず与えられた任意の正規式に対し，この正規式が表す言語をちょうど受理する ε 遷移を許す非決定性有限オートマトンで作ってみる．作り方は，構成要素の小さい部品から大きい部品へと帰納的に組み立てていけばよい．

方法 3.1 (正規式からの ε 遷移を許す非決定性有限オートマトンの構成)
入力: 正規式.
出力: 正規式の表す言語を受理する ε 遷移を許す非決定性有限オートマトン.

手順:

1. ε 遷移を許す非決定性有限オートマトンは，与えられた正規式の構成法に関する帰納法を利用して，構成することができる．構成法を表 3.5 に示す．ただし，式 e と式 e_1 の遷移表の開始記号を $0'$，終了状態を $1'$，式 e_2 の遷移表の開始状態を $0''$，終了状態を $1''$ と呼ぶことにする．□

例 3.1 (正規式からの構成)
正規式 a·a∗·b·b∗ を認識する有限オートマトンを構成してみる．まず式 a を認識する表 3.6 の遷移表を作る．同様に，正規式 b を認識する表 3.7 の遷移表を作る．次に，正規式 a∗ を認識する表 3.8 の遷移表を作る．そして，正規式 a·a∗ を認識する表 3.9 の遷移表を作る．最終的に，正規式 a·a∗·b·b∗ を認識する表 3.10 の状態遷移表を得る．□

表 3.5　状態遷移表の構成法

(1) 1 記号 a の場合の遷移表の構成法

状態	入力	状態
0	a	1(終了)

(2) 空列記号 ε の場合の遷移表の構成法

状態	入力	状態
0	ε	1(終了)

(3) 空集合記号 ϕ の場合の遷移表の構成法

状態	入力	状態
0		1

(4) 式 e_1 の遷移表と式 e_2 の遷移表が存在する場合の式 $(e_1)+(e_2)$ の遷移表の構成法

状態	入力	状態
0	ε	$0'$
0	ε	$0''$
$0'$	$\cdots$	
	式 e_1 の遷移表	
	$\cdots$	$1'$
$0''$	$\cdots$	
	式 e_2 の遷移表	
	$\cdots$	$1''$
$1'$	ε	1(終了)
$1''$	ε	1(終了)

(5) 式 e_1 の遷移表と式 e_2 の遷移表が存在する場合の式 $(e_1)\cdot(e_2)$ の遷移表の構成法

状態	入力	状態
$0'$	$\cdots$	
	式 e_1 の遷移表	
	$\cdots$	$1'$
$1'$	ε	$0''$
$0''$	$\cdots$	
	式 e_2 の遷移表	
	$\cdots$	$1''$(終了)

(6) 式 e の遷移表が存在する場合の式 $(e)*$ の遷移表の構成法

状態	入力	状態
0	ε	$0'$
0	ε	1(終了)
$0'$	$\cdots$	
	式 e の遷移表	
	$\cdots$	$1'$
$1'$	ε	$0'$
$1'$	ε	1(終了)

(7) 式 e の遷移表が存在する場合の式 (e) の遷移表は式 e の遷移表と同一とする．

表 3.6 状態遷移表

状態	入力	状態
0	a	1(終了)

表 3.7 状態遷移表

状態	入力	状態
0	b	1(終了)

表 3.8 状態遷移表

状態	入力	状態
0	ε	3(終了)
0	ε	1
1	a	2
2	ε	1
2	ε	3(終了)

表 3.9 状態遷移表

状態	入力	状態
0	a	1
1	ε	2
2	ε	5(終了)
2	ε	3
3	a	4
4	ε	3
4	ε	5(終了)

表 3.10 状態遷移表

状態	入力	状態
0	a	1
1	ε	2
2	ε	5
2	ε	3
3	a	4
4	ε	3
4	ε	5
5	ε	6
6	b	7
7	ε	8
8	ε	11(終了)
8	ε	9
9	b	10
10	ε	9
10	ε	11(終了)

ε 遷移を許す非決定性有限オートマトンから，完全定義型決定性有限オートマトンを作る手順を，方法 3.2 として示す．もともとの状態の集合を，1 つの新しい状態とするので，**部分集合構成法**と呼ぶ．もともとの状態 0 個からなる空集合は，新しいオートマトンのエラー状態として機能する．

定義 3.5 (ε 閉包)

q は ε 遷移を許す非決定性有限オートマトン M の状態で，**ε 閉包 (q)** とは，状態 q から 0 回以上の ε 遷移のみで到達できる M の状態すべての集合である． □

方法 3.2 (部分集合構成法)

入力: ε 遷移を許す非決定性有限オートマトン M.

出力: $\mathrm{L(M')} = \mathrm{L(M)}$ となる完全定義型決定性有限オートマトン M′.

手順:

1. M の状態の集合の集合を C で表し，$\mathrm{C} = \phi$ とする．次の状態 I_0

$$\mathrm{I}_0 = \varepsilon \text{ 閉包 } (i_0)$$

を "未処理" とマーク付けして，C に加える．ただし，i_0 は M の開始状態である．

2. C 内の未処理の状態を 1 つ選び，I と呼ぶ．状態 I を "既処理" とマーク付けする．
3. 入力記号の集合を $\{X_1, X_2, \cdots, X_n\}$ $(n > 0)$ とする．i を 1 から n まで以下を繰り返す．
   (i) 集合 $T_1 = \{\ q \mid j$ は I の要素で，q は $\delta(q, X_i)$ の要素 $\}$ を求め，T $= \varepsilon$ 閉包 (T_1) とする．
   T が C に既出の I′ と一致するのならば，GOTO(I, X_i) = I′ とする．
   T が C に既出の状態と一致しないのならば，新しい状態名 J と名付け，GOTO(I, X_i) = J とし，状態 J を "未処理" とマーク付けして，C に加える．
4. C 内の未処理の状態を 1 つ選び，I と呼ぶ．状態 I を "既処理" とマーク付けし，ステップ 3. へ行く．C 内に未処理の状態がなければ，ステップ 5. へ行く．
5. C を M′ の状態の集合とする．I_0 を M′ の開始状態とする．M の終了状態を少なくとも 1 つ含む C の状態を，M′ の終了状態とする．関数 GOTO を M′ の遷移関数とする．□

例 3.2 (決定性有限オートマトン)

ε 遷移を許す非決定性有限オートマトンである表 3.10の遷移表から，完全定義型決定性有限オートマトンである表 3.11の遷移表を次のように作ることができる．例えば，状態 0 の記号 a による遷移先は状態 1 のみであるが，状態 1 から ε 遷移を 1 回たどることにより，状態 2 に到達し，状態 2 から ε 遷移を 1 回たどることにより，状態 3 と状態 5 に到達し，状態 5 からさらに ε 遷移を 1 回たどることにより，状態 6 に到達する．したがって，GOTO({0}, a) = {1, 2, 3, 5, 6} となる．表 3.11 の遷移表の状態の名前を書き直すと表 3.12 の遷移表となる．□

定義 3.6 (状態の区別不能関係)

完全定義型の決定性有限オートマトンの状態集合の上に，長さ n 以下の記

表 3.11 状態遷移表

状態	入力	状態
{0}	a	{1, 2, 3, 5, 6}
{0}	b	{}
{1, 2, 3, 5, 6}	a	{3, 4, 5, 6}
{1, 2, 3, 5, 6}	b	{7, 8, 9, 11}(終了)
{3, 4, 5, 6}	a	{3, 4, 5, 6}
{3, 4, 5, 6}	b	{7, 8, 9, 11}(終了)
{7, 8, 9, 11}	a	{}
{7, 8, 9, 11}	b	{9, 10, 11}(終了)
{9, 10, 11}	a	{}
{9, 10, 11}	b	{9, 10, 11}(終了)
{}	a	{}
{}	b	{}

表 3.12 状態遷移表

状態	入力	状態
0	a	1
0	b	5
1	a	2
1	b	3(終了)
2	a	2
2	b	3(終了)
3	a	5
3	b	4(終了)
4	a	5
4	b	4(終了)
5	a	5
5	b	5

号列で**区別不能**という同値関係 $\underset{n}{\sim}$ を，次のように定義する．

$p \underset{n}{\sim} q \overset{定義}{\iff}$ 長さ n 以下のすべての記号列 w に対して，状態 p と q は，記号列 w を消費して，共に終了状態か，共に非終了状態に到達する． □

この関係は次のように帰納的に計算することができる．

1. まず，関係 $\underset{0}{\sim}$ を次のように定義する．

$q \underset{0}{\sim} q' \overset{定義}{\iff}$ q と q' が共に M の終了状態，または q と q' が共に M の非終了状態である．

2. 関係 $\underset{k}{\sim}$ を用いて，関係 $\underset{k+1}{\sim}$ を求める．

$q \underset{k+1}{\sim} q' \overset{定義}{\iff}$ $q \underset{k}{\sim} q'$ でかつすべての入力記号 X について，$\delta(q, \mathrm{X}) \underset{k}{\sim} \delta(q', \mathrm{X})$．

完全定義型の決定性有限オートマトンから状態数最小の完全定義型の決定性有限オートマトンを作る手順を方法 3.3 として示す．これで，与えられた正規式のパターンを認識する状態数最小の完全定義型の決定性有限オートマトンが構成できたことになる．

方法 3.3 (完全定義型決定性有限オートマトンの最小化)

入力: 完全定義型の決定性有限オートマトンM.

出力: L(M) = L(M′) で，状態数が最小の完全定義型の決定性有限オートマトン M′.

手順:

1. M より，初期状態から到達不可能な状態を除去した結果を，M_0とする．
2. M_0 の状態集合上の同値関係 $\underset{0}{\sim}$ から始めて，次々と$\underset{1}{\sim}$，$\underset{2}{\sim}$，$\cdots$ を計算する．$\underset{t}{\sim}$ と $\underset{t+1}{\sim}$ が同一となったところで終了する．(つまり M_0 の状態集合を非終了状態と終了状態の 2 つの派閥に分割する．派閥に属す各状態の入力記号に対する遷移先の派閥名を調べる．同一派閥内の状態の遷移先派閥名のふるまいが同一なら，特に何もしない．ふるまいが同一でないときは，ふるまいの同一ないくつかの派閥へとその派閥を分割する．各派閥内のふるまいがそれぞれ同一になったところで終了する.)
3. $\underset{t}{\sim}$ で同一視できる状態を状態の集合とするオートマトン M_1 を作る．
4. もともとの開始状態を含む状態を，開始状態とする．もともとの終了状態を含む状態を，終了状態とする．M_1を M′ とする．遷移関数が部分関数でもよければ，さらに M_1 より，終了状態へ到達不可能な状態を除去した結果を M′ とする．□

例 3.3 (状態数の最小化)

表 3.12 の完全定義型の決定性有限オートマトンから，状態数が最小の決定性有限オートマトンを計算する過程は表 3.13 のようになる．関係 $\underset{0}{\sim}$ を示す状態集合の 2 分割からスタートして，関係 $\underset{1}{\sim}$ を示す 3 分割，関係 $\underset{2}{\sim}$ を示す 4 分割，関係 $\underset{3}{\sim}$ を示す 4 分割と作っていくと，関係 $\underset{3}{\sim}$ は関係 $\underset{2}{\sim}$ と一致することがわかる．すなわち，非終了状態と終了状態の 2 分割では，派閥 A に属す状態のふるまいが AA 型と AB 型の 2 種類あるため，派閥 A を派閥 C と D へ分割することが必要となる．次の 3 分割では派閥 D に属す状態のふるまいが CD 型と DD 型の 2 種類あるため，やはり派閥 D の派閥 E と F への分割が必要となる．そして， 4 分割では各派閥内のふるまいは同一となるので，終了する．最小状態数は 4 となり，最小状態数の完全定義型決定性有限オートマトンが得られる．結局，表 3.14 の遷移表が得られる．

表 3.13　最小化の過程

A:{0, 1, 2, 5}, B:{3, 4}

	0	1	2	5	3	4
a	A	A	A	A	A	A
b	A	B	B	A	B	B

D:{0, 5}, C:{1, 2}, B:{3, 4}

	0	5	1	2	3	4
a	C	D	C	C	D	D
b	D	D	B	B	B	B

F:{0}, E:{5}, C:{1, 2}, B:{3, 4}

	0	5	1	2	3	4
a	C	E	C	C	E	E
b	E	E	B	B	B	B

表 3.14　状態遷移表

A:{0, 1, 2, 5}, B:{3, 4}

状態	入力	状態
0	a	1
0	b	3
1	a	1
1	b	2(終了)
2	a	3
2	b	2(終了)
3	a	3
3	b	3

表 3.15　状態遷移表

状態	入力	状態
0	a	1
1	a	1
1	b	2(終了)
2	b	2(終了)

遷移関数が部分関数でもよければ，終了状態へ到達しない状態を除いて，決定性有限オートマトンは表 3.15 のようになり，状態数は 3 となる．　□

3.3　直接的構成法

正規式から直接的に最小状態数の完全定義型決定性有限オートマトンを構成する手順を示す．

定義 3.7 (正規式の微分)

正規式 e に対し，次のような正規式 f を，記号列 x による正規式 e の**微分** D[x](e) と呼ぶ．

$$\mathrm{L}(f) = \{\ \mathrm{y} \mid \mathrm{xy}\ \text{は}\ \mathrm{L}(e)\ \text{の要素}\ \}.$$ □

記号列 x による正規式 e の微分 $\mathrm{D[x]}(e)$ は，次のように帰納的に計算することができる．以下で a, b は 1 記号で，e, e_0, e_1, e_2 は正規式とする．

1. $\mathrm{D[ax]}(e) = \mathrm{D[x]}(\mathrm{D[a]}(e))$,
   $\mathrm{D}[\varepsilon](e) = e$.
2. $\mathrm{D[a]}(\varepsilon) = \phi$,
   $\mathrm{D[a]}(\phi) = \phi$,
   $\mathrm{D[a](a)} = \varepsilon$,
   $\mathrm{D[a](b)} = \phi$ $(\mathrm{a} \neq \mathrm{b})$,
   $\mathrm{D[a]}(e_1 + e_2) = \mathrm{D[a]}(e_1) + \mathrm{D[a]}(e_2)$,
   $\mathrm{D[a]}(e_0 \cdot e_2) = \mathrm{D[a]}(e_0) \cdot e_2 + \mathrm{D[a]}(e_2)$ (ε が $\mathrm{L}(e_0)$ の要素である場合),
   $\mathrm{D[a]}(e_1 \cdot e_2) = \mathrm{D[a]}(e_1) \cdot e_2$ (ε が $\mathrm{L}(e_1)$ の要素でない場合),
   $\mathrm{D[a]}(e*) = \mathrm{D[a]}(e) \cdot e*$.

さらに，一般の正規式 e に対し，$\mathrm{L}(e)$ が ε を要素に持つ正規式であるかを判定する述語 ε 判定 (e) は，次のように帰納的に定義できる．f_1, f_2 は一般の正規式とする．

ε 判定 (ε) = 真,
ε 判定 (ϕ) = 偽,
ε 判定 (a) = 偽,
ε 判定 $(f_1 + f_2)$ = ε 判定 (f_1) **or** ε 判定 (f_2),
ε 判定 $(f_1 \cdot f_2)$ = ε 判定 (f_1) **and** ε 判定 (f_2),
ε 判定 $(e*)$ = 真,
ε 判定 $((e))$ = ε 判定 (e).

方法 3.4 (正規式からの直接的構成法)
入力: 正規式 e.
出力: 言語 $\mathrm{L}(e)$ を受理する最小状態数の完全型決定性有限オートマトン M.
手順:
1. 正規式の集合を C で表し，C = {} とする．C はこれから作るオートマトンの状態の集合である．正規式 e を “未処理” とマーク付けして，C

に加える．

2. C 内の未処理の正規式を 1 つ選び，I と呼ぶ．正規式 I を “既処理” とマーク付けする．
3. 入力記号の集合を $\{X_1, X_2, \cdots, X_n\} (n \geq 1)$ とする．i を 1 から n まで以下を繰り返す．
   (i) $D[X_i](I)$ を計算し，T とする．T が C に既出の I' に対し，$L(I') = L(T)$ となるならば，$\delta(I, X_i) = I'$ とする．そうでなければ，$\delta(I, X_i) = T$ とし，正規式 T を “未処理” とマーク付けして，C に加える．
4. C 内の未処理の状態を 1 つ選び，I と呼ぶ．状態 I を “既処理” とマーク付けし，ステップ 3. へ行く．C 内に未処理の状態がなければ，ステップ 5. へ行く．
5. C の正規式のうち，$L(f)$ が ε を持つ正規式 f を，すべて終了状態とする． □

2 つの正規式 e と f について，$L(e) = L(f)$ をチェックすることは一般的に可能ではあるが，かなりの手間となる．すぐに $L(e) = L(f)$ がわからない場合には，違うものとしておけば，状態数が増加することはあるが，正しい決定性有限オートマトンが得られる．

例 3.4 (正規式からの直接的構成法)

正規式 a·a∗·b·b∗ から直接オートマトンを構成してみる．

1. 正規式 a·a∗·b·b∗ を a で微分すると，正規式 a∗·b·b∗ となり，b で微分すると，正規式 ϕ となる．
2. 正規式 a∗·b·b∗ を a で微分すると，正規式 a∗·b·b∗ となり，b で微分すると，b∗ となる．
3. 正規式 b∗ を a で微分すると，正規式 ϕ となり，b で微分すると，b∗ となる．
4. 正規式 ϕ を a で微分すると，正規式 ϕ となり，b で微分すると，正規式 ϕ となる．

開始状態は a·a∗·b·b∗ で，終了状態は b∗ である．以上から得られる最小状態数の完全定義型決定性有限オートマトンは表 3.16 の遷移表となる． □

表 3.16　状態遷移表

状態	入力	状態
0	a	1
0	b	3
1	a	1
1	b	2(終了)
2	a	3
2	b	2(終了)
3	a	3
3	b	3

3.4　字句解析系

実際の字句解析では，字句認識と異なり，長い文字列から 1 つずつ字句記号を取り出さなければならない．したがって，すべての字句記号のパターンの受理言語の和集合を受理する決定性有限オートマトンを作り，はじめてすべてが “エラー” となった時刻から考えて，最も最近に “yes” と出力した状態を真の終了状態としなければならない．そしてそのような真の受理状態が 2 種類以上の字句記号に対して，“yes” と出力している場合は，例えば字句記号としての番号の若いほうの字句記号を選べばよい．

例えば，字句 1 のパターンが a·b*，字句 2 のパターンが a·b·c のとき，文字列 abcabb を区切る場合を考えてみよう．表 3.17 の遷移表を準備して，開始状態から動作させてみる．開始状態 0 から 1 文字目 a を消費した時点で字句 1 と認識し， 2 文字目 b を消費した時点で同じく字句 1 と認識し，3 文字目 c を消費した時点で字句 2 と認識し，4 文字目 a を消費した時点でエラーと認識する．そこで，1 文字目から 3 文字目までを字句 2 として切り出すことになる．残りの列 abb についても同様で，開始状態 0 から 4 文字目 a を消費した時点で字句 1 と認識し，5 文字目 b を消費した時点で同じく字句 1 と認識し，6 文字目 b を消費した時点でも同じく字句 1 と認識し，文字列が尽きてエラーとなる．これで 4 文字目から 6 文字目までを字句 1 として切り出すことができる．

表 3.17　状態遷移表

状態	入力	状態
0	a	1(字句 1 の認識)
0	b, c	5
1	b	2(字句 1 の認識)
1	a, c	5
2	b	3(字句 1 の認識)
2	c	4(字句 2 の認識)
2	a	5
3	b	3(字句 1 の認識)
3	a, c	5
4	a, b, c	5
5	a, b, c	5

字句解析系を生成するシステムの例としては，AT&T の LEX システムがある．このシステムへの入力の例をプログラム 3.4 に 1 つ示す．プログラム 3.4 では，空白とタブ記号を捨て，字句記号 IDENT と NUMBER を定義し，それ以外の 1 記号を 1 字句記号と定義している．

プログラム 3.4

```
%%
[ \t]              {;}
[a-z][a-z0-9]*     {return IDENT;}
[0-9]+             {return NUMBER;}
\n|.               {return yytext[0];}
```

第4章　字句解析の理論

パターンを表す正規式，記号列を認識する有限オートマトン，正規言語を生成する正規文法の関係について述べ，これらは同一の言語クラスを定義していることを示す．最後に最小状態数の有限オートマトンについての議論を示す．

4.1　正規文法と正規言語

まず，正規式が表す言語と有限オートマトンが受理する言語のクラスの同一性について述べる．

定理 4.1 (正規式で表す言語)
入力記号集合 T 上の正規式で表される言語を，ちょうど受理する T 上の完全定義型の決定性有限オートマトン M が存在する．□

証明
第 3 章で示した間接的構成法により，正規式で表された言語をちょうど受理する完全定義型の決定性有限オートマトンをいつも構成することができる．これで題意が示せた．また間接的構成法の議論により，ε 遷移を許す非決定性有限オートマトンの受理言語のクラス，非決定性有限オートマトンの受理言語のクラス，完全定義型の決定性有限オートマトンの受理言語のクラスは同一であることも示せた．□

定理 4.2 (受理言語)
入力記号集合 T の上の非決定性有限オートマトン M の受理言語 L(M) は，T 上の正規式で表すことができる．　□

証明
非決定性有限オートマトン M の状態遷移図は，ラベル付き有向グラフ G = (V, E) である．ここで，V は状態の集合 $\{q_1, q_2, \cdots, q_n\}$ で，E は (q_i, a, q_j) の形をしたラベル付き有向枝の集合である．ラベル付き有向グラフ G の任意の 2 節点 q_i と q_j の間の有限の長さの経路の全体 $\mathrm{T}(i, j)$ が，ラベル集合の上の正規式で表せることを示せば十分である．なぜならば受理言語は，開始状態から終了状態へ至る経路の集合の有限個の和集合として表せるからである．

$\mathrm{T}(i, j)$ は次のように帰納的に求めることができる．ここで，$\mathrm{T}[k](i, j)$ は，節点 q_i から出発して，途中 $\{q_1, q_2, \cdots, q_k\}$ の範囲内で寄り道をしたり，しなかったりして，節点 q_j に到着する有限の長さの経路の全体である．

1. $k = 0$ で $i = j$ のとき，集合で表現すると，

$$\mathrm{T}[0](i, j) = \{\ \mathrm{a} \mid (q_i, \mathrm{a}, q_j) \text{ は E の要素 }\} \cup \{\varepsilon\}.$$

したがって，

$$\mathrm{T}[0](i, j) = \mathrm{a}_{01} + \cdots + \mathrm{a}_{0m} + \varepsilon$$

と正規式で表せる．

$k = 0$ で $i \neq j$ のとき，集合で表現すると，

$$\mathrm{T}[0](i, j) = \{\ \mathrm{a} \mid (q_i, \mathrm{a}, q_j) \text{ は E の要素 }\}.$$

したがって，

$$\mathrm{T}[0](i, j) = \mathrm{a}_{01} + \cdots + \mathrm{a}_{0m}$$

と正規式で表せる．

2. $k > 0$ のとき，以下の関係が成立する．

$$\mathrm{T}[k](i, j) = \mathrm{T}[k-1](i, j) + \mathrm{T}[k-1](i, k)\cdot(\mathrm{T}[k-1](k, k))*\cdot\mathrm{T}[k-1](k, j)$$

(ここで右辺は，節点 q_k を有限回訪れる経路を，ちょうど 0 回訪れる経路，ちょうど 1 回訪れる経路，ちょうど 2 回訪れる経路，$\cdots$，と場合分けしている.)

したがって，

$\mathrm{T}[k-1](i,\, j),\ \mathrm{T}[k-1](i,\, k),\ \mathrm{T}[k-1](k,\, k),\ \mathrm{T}[k-1](k,\, j)$

が正規式ならば，

$\mathrm{T}[k](i,\, j)$

も正規式で表せる．

3. $\mathrm{T}(i,\, j) = \mathrm{T}[n](i,\, j)$ で計算が終了し，経路の全体が正規式で表せた．□

以上から終端記号 T 上の正規式で表せる言語のクラスは，入力記号の集合 T の上の有限オートマトンの受理する言語のクラス（ε 遷移を許す非決定性有限オートマトンの受理言語のクラス，非決定性有限オートマトンの受理言語のクラス，完全定義型の決定性有限オートマトンの受理言語のクラス）と一致することが示された．

定理 4.2 の証明の副産物として，有向グラフに関する 2 つの重要なアルゴリズム，最短経路コストの計算法と到達可能性の判定法が得られるので示しておく．有向グラフ $\mathrm{G} = (\mathrm{V},\, \mathrm{E})$ は，節点集合 $\mathrm{V} = \{1,\, 2,\, \cdots,\, n\}$，$i \neq j$ である有向枝 $(i,\, j)$ に正のコスト C_{ij} が定義されているものとする．

1. 最短経路コストの計算法

G 上のすべての 2 節点 $i,\, j$ 間の最短経路コスト $\mathrm{T}(i,\, j)$ は，次の帰納的計算法で計算できる．まず，$\mathrm{T}[0](i,\, j)$ を次のように定義する．

$$\mathrm{T}[0](i,\, j) = \begin{cases} 0 & (i = j), \\ \mathrm{C}_{ij} & (i \neq j \text{ なる有向枝 } (i,\, j) \text{ が存在するとき}), \\ \infty & (i \neq j \text{ なる有向枝 } (i,\, j) \text{ が無いとき}). \end{cases}$$

そして，$k \geq 1$ の $\mathrm{T}[k](i,\, j)$ を次のように定義する．

$$\mathrm{T}[k](i,\, j) = MIN(\mathrm{T}[k-1](i,\, j), \mathrm{T}[k-1](i,\, k) + \mathrm{T}[k-1](k,\, j)).$$

最後に，$\mathrm{T}(i,\, j) = \mathrm{T}[n](i,\, j)$ で最短経路のコストが求まる．

2. 到達可能性の判定法

G 上のすべての 2 節点 i, j 間の到達可能性は，次の帰納的計算法で判定できる．まず $\mathrm{T}[0](i,\, j)$ を次のように定義する．

$$\mathrm{T}[0](i,\,j) = \begin{cases} 1 & (i = j), \\ 1 & (i \neq j \text{ なる有向枝 } (i,\,j) \text{ が存在するとき}), \\ 0 & (i \neq j \text{ なる有向枝 } (i,\,j) \text{ が無いとき}). \end{cases}$$

そして，$k \geq 1$ の $\mathrm{T}[k](i,\,j)$ を次のように定義する．

$$\mathrm{T}[k](i,\,j) = MAX(\mathrm{T}[k-1](i,\,j), \mathrm{T}[k-1](i,\,k) \times \mathrm{T}[k-1](k,\,j)).$$

最後に，$\mathrm{T}(i,\,j) = \mathrm{T}[n](i,\,j)$ で到達可能性が判定できる．値 1 は到達可能，値 0 は到達不能であることを示している．

次に正規言語と有限オートマトンの受理言語の関係を述べる．

定理 4.3 (正規言語と有限オートマトンの受理言語)

1. 終端記号の集合 T の正規文法 G で生成できる言語 L は，入力記号の集合 T の ε 遷移を許す非決定性有限オートマトン M で受理できる．
2. 入力記号の集合 T の ε 遷移を許す非決定性有限オートマトン M の受理言語 L は，終端記号の集合 T の正規文法 G で生成できる．　□

証明

1. 正規文法 G = (N, T, P, S) の各生成規則から，次のような ε 遷移を許す非決定性有限オートマトン M = (N ∪ $[q]$, T, δ, S, F) を作る．ただし，F = {$[q]$} とする．

   A → a B から，状態 B を δ(A, a) の要素とする．

   A → a から，状態 $[q]$ を δ(A, a) の要素とする．

   そして，S → ε が存在するならば，状態 $[q]$ を δ(S, ε) の要素とする．

2. ε 遷移を許す非決定性有限オートマトン M = (Q, T, δ, i, F) の各状態遷移から，正規文法 G = (Q, T, P, i) の生成規則を次のように作る．

   状態 r が δ(p, a) の要素なら，生成規則 p → a r を P の要素とする．

   状態 r が δ(p, ε) の要素なら，生成規則 p → r を P の要素とす

る．

さらに，r が終了状態集合 F の要素ならば，生成規則 $r \to \varepsilon$ を P の要素とする．□

なお，完全定義型の決定性有限オートマトンの終了状態と非終了状態を入れ替えると，ちょうどその言語の補集合が受理言語となる．(非決定性有限オートマトンの場合は，終了状態と非終了状態を入れ替えても，必ずしも言語の補集合が受理言語とはならない点に注意されたい.) この事実と，集合 A，B についての以下の De Morgan の法則

$$(A^c \cup B^c)^c = A \cap B$$

から，正規言語に対して和集合操作，補集合操作，積集合操作を行った結果は，すべて正規言語となることがわかる．

4.2　正規言語の特徴付け

部屋が n 部屋の旅館に，$n+1$ 人以上の客を泊める場合，2 人以上の客のいる部屋が少なくとも 1 つ存在するという論法を，**部屋割り論法**という．この部屋割り論法を用いて，正規言語についてのポンプの補題を示す．

定理 4.4 (正規言語のポンプの補題)
正規言語 L に対して，ある整数 k が存在して，長さ k 以上のどんな L の文 z も，次の性質を満たす．

1. z = wxy と書ける．
2. x の長さは 1 以上 k 以下である．
3. すべての 0 以上の整数 i に対し，wx^iy は L の文である．□

証明
正規言語 L を受理する決定性有限オートマトンの 1 つ M を考える．M の状態の数を k とする．長さ k 以上の L の文がそもそも存在しない場合には，前提が偽となるので題意全体が真となり成立する．そこで長さ k 以上の L の文が 1 個以上存在する場合を扱う．長さ k 以上の記号列 $X_1X_2\cdots X_n$

$(n \geq k)$ を考えると，L の文なので，受理までに開始状態 s_0 と最終状態 s_n を含めて，合計 $n+1$ 個の状態を以下の順番に通過する．

$$s_0, s_1, \cdots, s_n$$

$n+1$ は状態の個数 k より大きいので，ある s_i と s_j $(0 \leq i < j \leq k \leq n)$ の対が少なくとも存在して，同じ状態となる．(そうでなければ，$k+1$ 個でなく k 個分までしか状態を通過できない.)

そこで s_0 から s_i で消費した記号列を w，s_i から s_j で消費した記号列を x，s_j から s_n で消費した記号列を y とおく．$0 \leq i < j \leq k$ から，x の長さは 1 以上 k 以下である．また $s_i = s_j$ なので，wy，wxy，wxxy，wxxxy，$\cdots$ はすべて M で受理できる．□

定理 4.5 (非正規言語)

$L = \{a^n b^n \mid n$ は 1 以上の整数 $\}$ は，正規言語でない．□

証明

もし L が正規言語であるとすると，定理 4.4 の性質を満たす整数 kが存在する．そこで，長さ k 以上である文 $a^k b^k$ を考える．$a^k b^k$ は wxy と書けることになる．x の長さは 1 以上 k 以下である．wy も wxy も L の要素で，それぞれ同じ個数の a と b を含み，wxy の長さは wy より大きい．したがって，x には 1 個以上の a と 1 個以上の b が同数含まれる．wxxy も L の要素である．wxxy の xx の部分には，a と a の間に少なくとも 1 個の b が存在する．よって wxxy は L の要素でない．これは矛盾である．したがって L は正規言語ではない．□

別証

正規言語とすると，L を受理する決定性有限オートマトン M が存在する．M の状態の数を k とする．$k+1$ 個の入力列

$$ab, a^2b^2, \cdots, a^kb^k, a^{k+1}b^{k+1}$$

を考える．状態は k 個しかないので，ある i と j $(1 \leq i < j \leq k)$ が存在して，開始状態から，a^i を消費して到達する状態と，a^j を消費して到達する

状態は一致してしまう．この状態を q と呼ぶ．状態 q は b^i を消費してある終了状態に到達する．同様に状態 q は b^j を消費してどれかの終了状態に到達する．したがって開始状態から $\mathrm{a}^i\mathrm{b}^j$ を消費して終了状態に到達する．これは受理言語が $\{\mathrm{a}^n\mathrm{b}^n \mid n$ は 1 以上の整数$\}$ であることに矛盾する．　□

同様に状態数 n の有限オートマトン M に対して，部屋割り論法により，長さ n 以上の受理された入力記号列 x が存在するならば，x より長さが 1 以上短くてしかも受理できる入力記号列 y が作れることがわかる．この結果，次の性質が成り立つ．

1. 受理言語 L(M) が空集合であるための必要十分条件は，長さ 0 から長さ $n-1$ までの範囲の入力記号列を，1 つも受理しないことである．

同様に状態数 n の有限オートマトン M に対して，長さ $2n$ 以上の受理された入力記号列 x が存在するならば，x より長さが 1 以上短くてしかも同一の状態を 2 回以上通過して受理される入力記号列 y が作れることがわかる．したがって，次の性質も成り立つ．

2. 受理言語 L(M) が無限集合であるための必要十分条件は，長さ n から長さ $2n-1$ までの範囲の入力記号列を，少なくとも 1 つ受理することである．

4.3　最小状態有限オートマトン

最小状態数の有限オートマトンの性質を調べる．

定義 4.1 (状態の区別不能関係)
完全定義型の決定性有限オートマトン M の状態集合の上に，どんな長さの記号列でも区別不能という同値関係 $\sim$ を，次のように定義する．

$p \sim q \overset{\text{定義}}{\iff}$ どんな記号列 w に対しても，状態 p と q は記号列 w を消費して共に最終状態か，共に非最終状態に到達する．　□

定義 4.2 (記号列の区別不能関係)
完全定義型の決定性有限オートマトン M のすべての状態が，開始状態から到達可能なとき，入力記号の有限列の全体 T* の上に，同値関係 ∼ を次のように定義する．

$x \sim y \overset{\text{定義}}{\Longleftrightarrow}$ どんな記号列 w に対しても，xw と yw は共に L(M) の要素，あるいは xw と yw は共に T* − L(M) の要素． □

以上の 2 つの同値関係には次のような関係がある．

完全定義型の決定性有限オートマトン M のすべての状態が，開始状態から到達可能なとき，記号列 x が開始状態から状態 p に到達し，y が開始状態から状態 q に到達するとすれば，$p \sim q$ と x ∼ y は同値となる．

定理 4.6 (受理言語の同一性)
方法 3.3 のステップ 3. の出力として得られる，入力記号集合 T の上の決定性有限オートマトンを $\mathrm{M_1} = (\mathrm{Q_1}, \mathrm{T}, \delta_1, i_1, \mathrm{F_1})$，$\mathrm{M_1}$の受理する言語を $\mathrm{L_1}$ とする．同ステップ 2. の入力となる，入力記号集合 T の上の非決定性有限オートマトンを $\mathrm{M_0} = (\mathrm{Q_0}, \mathrm{T}, \delta_0, i_0, \mathrm{F_0})$，$\mathrm{M_0}$ が受理する言語を $\mathrm{L_0}$ とする．$\mathrm{L_1} = \mathrm{L_0}$ である． □

証明
T* の上の同値関係 ∼ の同値類の集合を $\{q_1, q_2, \cdots, q_n\}$ とし，記号列 x の属す同値類を ≪x≫ とする．構成法から，$\mathrm{Q_1} = \{q_1, q_2, \cdots, q_n\}$，≪x≫ $= q_i$ で，≪xa≫ $= q_j$ のとき，$\delta_1(q_i, \mathrm{a}) = q_j$，$i_1 = \ll \varepsilon \gg$，$\mathrm{F_1} = \{\, q \mid$ w は L の要素，≪w≫ $= q\}$ となる．

$\mathrm{M_0}$ の上で，i_0 が w を消費して ($\mathrm{Q_0} - \mathrm{F_0}$の要素である) 非終了状態に到達するなら，すなわち w が T* − $\mathrm{L_0}$ の要素なら，$\mathrm{M_1}$ の上で，i_1 が w を消費して状態 ≪w≫ に到達し，≪w≫ は $\mathrm{F_1}$ の要素でない．

同様に $\mathrm{M_0}$ の上で，i_0 が w を消費して ($\mathrm{F_0}$ の要素である) 終了状態に到達するなら，すなわち w が $\mathrm{L_0}$ の要素なら，$\mathrm{M_1}$ の上で，i_1 が w を消費して状態 ≪w≫ に到達し，≪w≫ は $\mathrm{F_1}$ の要素となる．以上から題意が成立

する. □

定理 4.7 (状態数の最小性)
方法 3.3 で構成した入力記号集合 T の上の決定性有限オートマトンを M_1, M_1 が受理する言語を L_1 とする. L_1 を受理する, 入力記号集合 T の上の任意の全関数の遷移関数を持つ決定性有限オートマトンを M_2 とする. M_2 のどの状態も開始状態から到達可能なら, M_2 の状態数は, M_1 の状態数以上である. □

証明
T∗ 上の 2 つの同値関係 R_1 と R_2 を考える.

$$x\, R_1\, y \overset{\text{定義}}{\Longleftrightarrow} x \sim y.$$

$$x\, R_2\, y \overset{\text{定義}}{\Longleftrightarrow} x \text{ と } y \text{ は開始状態から } M_2 \text{ の同一状態に到達する.}$$

このとき, $x\, R_2\, y$ ならば必ず $x\, R_1\, y$ である. なぜなら, 同一状態に到達した 2 つの記号列は, あと w を加えて同一の M_2 の状態に到達し, そこは最終状態か非最終状態のどちらかであるからである.

したがって, R_2 のどの 1 つの同値類も, 適当な R_1 の同値類 1 つの部分集合となっている. つまり, T∗ の R_2 による同値分割の数は, R_1 による同値分割の数以上である. T∗ の R_2 による同値分割の数は M_2 の状態数に等しい. T∗ の R_1 による同値分割の数は M_1 の状態数に等しい. よって題意は成立する. □

以上から, 方法 3.3 は, 受理言語が同一で, 状態数が最小の完全定義型決定性有限オートマトンを構成することが示せた.

第5章　構文解析の方法

本章では，まず構文解析を行うのに必要な概念と基本的方法を述べ，次に下降型構文解析，上昇型構文解析，混合型構文解析の代表的方法について述べる．最後に，手続き型構文解析と索表計算型構文解析について述べる．

5.1　構文解析の原理

構文解析は，与えられた入力列と文法に対して，その終端記号列の導出の過程を再現し，構文構造を決定するものである．したがって，再現の仕方で，構文解析法を 3 つに分類することができる．開始記号から出発して，終端記号列を得ようとする**下降型** (top-down) **解析法**，終端記号列から出発して，開始記号に到達しようとする**上昇型** (bottom-up) **解析法**，そして下降型と上昇型の**混合型解析法**である．

これらの構文解析の過程は，次の 3 つの構成要素を用いて記述できる．

1. 入力列

   当初，入力列は与えられた終端記号列の全体を保持している．構文解析の進行につれて照合済みの部分が増加し，未照合の部分が減少する．未照合部分の先頭 k 個の記号の列を，長さ k の**先読み** (lookahead) と呼ぶ．未照合部分は，照合が行われるにつれて，消滅していく．入力列の右端を示す記号が必要なときは，記号 \$ を用いることにする．

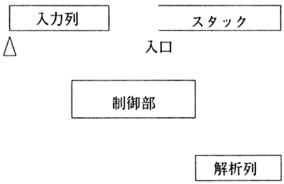


図 5.1　下降型解析のスタック

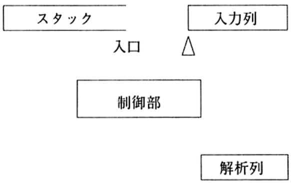


図 5.2　上昇型解析のスタック

2. スタック
最後に格納したものが最初に取り出せる記憶構造を，**スタック**と呼ぶ．スタックに格納することを，**プッシュ**といい，スタックから取り出すことを，**ポップ**という．当初，スタックの内容は構文解析の開始状況を示す記号または記号列が入っている．構文解析の進行につれて，スタックに対し記号をプッシュしたり，ポップする．照合済みの入力列部分の解析に対応して，スタックの内容は変化する．下降型解析では，図 5.1 のような左に入口のある横置きのスタック，上昇型解析では，図 5.2 のような右に入口のある横置きのスタックを用いる．

3. 解析列
当初，解析列は空列である．文法規則を認識するにつれて，文法規則の番号を出力して行く．構文解析が正しく終了すれば，正常終了を示す記号 a を出力して停止する．入力列に誤りを検出した場合は，異常終了を示す記号 e あるいはエラーメッセージを出力して停止する．

それでは，まず素朴な下降型解析法を 1 つ示す．

方法 5.1 (素朴式下降型解析)

入力: 文法規則の右辺の最左記号が必ず終端記号で，しかも同一左辺を持つ文法規則の最左記号がすべて異っているような文脈自由文法 G と終端記号列 x.

出力: 終端記号列 x の左解析列またはエラーメッセージ.

表 5.1 解析過程

入力列	スタック	解析列
ac	S	
ac	aB	1
c	B	1
c	c	15
		15a

手順:

1. 空スタックの中に開始記号 S を入れる．入力列の先頭 1 記号を先読み記号とする．
2. 以下をそれ以上繰り返せなくなるまで，繰り返す．
   (i) **(展開)** スタックの先頭記号が非終端記号 A なら，入力列の先読み記号 X ではじまる右辺を持つ文法規則 A $\rightarrow$ X α を選ぶ．規則番号を解析列に出力する．スタックの先頭部分の記号 A をポップし，右辺 X α の左右反転列を 1 記号ずつプッシュする．
   (ii) **(照合)** スタックの先頭記号が終端記号 X なら，先読み記号と等しいかどうか調べ，等しくなければエラーメッセージを出力して終了，等しければ，スタックおよび入力列から，それぞれ一致した記号 X を消去する．
   (iii) **(終了)** スタックの内容と入力列が共に空なら，解析列に a を出力して正常終了．入力列のみ非空なら，エラーメッセージを出力して終了． □

例 5.1 (素朴式下降型解析)
次の文法の場合，表 5.1 のような解析過程となる．

1) S $\rightarrow$ a B	4) A $\rightarrow$ d
2) S $\rightarrow$ b A	5) B $\rightarrow$ c
3) A $\rightarrow$ c	6) B $\rightarrow$ d

□

素朴式の下降型解析法は制限を持っているが，決定型の下降型構文解析の基本的アイデアを理解するのに役立つ．この方法を後で一般化するために，

展開時の右文脈という概念を定義しておく.

定義 5.1 (規則 A→ $\boldsymbol{\alpha}$ の展開時の長さ $\boldsymbol{k}$ の右文脈)
文法 G の規則 A $\rightarrow \alpha$ に対し，次のような最左導出において

$$\mathrm{S\$}^k \stackrel{*}{\Longrightarrow} \mathrm{xA}\beta \Rightarrow \mathrm{x}\alpha\beta \stackrel{*}{\Longrightarrow} \mathrm{xy}$$

終端記号列 y の $\mathrm{FIRST}_k(\mathrm{y})$ を，**規則 A $\rightarrow$ $\boldsymbol{\alpha}$ の展開時の長さ $\boldsymbol{k}$ の右文脈**という. □

次に素朴な上昇型解析法を示す.

方法 5.2 (素朴式上昇型解析)

入力: 文脈自由文法 G と終端記号列 x. ただし, 0 番目の文法規則は拡大文法とするために追加された規則 S′→S とする.

出力: 終端記号列 x についての必ずしも正しくない右解析列またはエラーメッセージ.

手順:

1. スタックの内容を空とする.
2. 次の (i) から (iv) のいずれかを選びながら，正常終了か異常終了になるまで繰り返す.
   - (i) (**シフト**) スタックの先頭部分にどの規則の右辺部分も出現していないのならば，入力列の先頭 1 記号をスタックにプッシュする.
   - (ii) (**還元**) スタックの先頭部分に i 番目の規則 ($i > 0$) の右辺が出現しているならば，右辺部分をポップし，i 番目の規則の左辺をプッシュする. 解析列に i を出力する.
   - (iii) (**正常終了**) 入力列が尽きて，スタックの先頭部分に 0 番目の規則の右辺が出現しているならば，解析列に a を出力し停止する.
   - (iv) (**異常終了**) 入力列が尽きて，しかもスタックの先頭部分に規則の右辺部分が出現していないならば，エラーメッセージを出力して終了する. □

表 5.2 解析過程

スタック	入力列	解析列
	aaa	
a	aa	
A	aa	3
Aa	a	3
A	a	32
Aa		32
A		322
S		3221
S′		3221a

例 5.2 (素朴式上昇型解析)
次の文法の場合，表 5.2 のような解析過程となる．

0) $S' \to S$　　2) $A \to A\ a$

1) $S \to A$　　3) $A \to a$

□

この素朴式上昇型解析法は一般には正しくない構文解析法である．最初にスタックの先頭部分に出現した右辺を左辺に還元していては，正しい解析結果が保証できない．しかしながら，素朴式上昇型解析法も決定型の上昇型構文解析の基本的アイデアを理解するのに役立つ．後で一般化するために，還元時の左文脈という概念を定義しておく．

定義 5.2 (規則 $\mathbf{A \to \alpha}$ の還元時の左文脈)
文法 G の拡大文法 G′ の規則 $A \to \alpha$ を用いる以下の最右導出において，

$$S'\$^k \stackrel{*}{\Longrightarrow} \beta A x \Rightarrow \beta\alpha x \stackrel{*}{\Longrightarrow} wx$$

$\beta\alpha$ を，**規則 $\mathbf{A \to \alpha}$ の還元時の左文脈**という．規則 $A \to \alpha$ の還元時の左文脈は，ちょうど規則 $A \to \alpha$ で還元するときのスタックの内容である．□

最後に，書換えの列の長さの広義の単調増加性を利用する単調増加法を示す．

方法 5.3 (単調増加法)

入力: 右辺長が左辺長以上となっている文脈自由文法 G と終端記号列 x.

出力: 終端記号列 x の左解析列，またはエラーメッセージ.

手順:

1. 文法 G の 0 番目の規則を $S' \to S$ とする．集合 M を文形式 S とその左解析列 P との対 (S, P) の集合とする．最初，集合 $M = \{(S, 0)\}$ とする.
2. ステップ (i) を，新しい要素が加えられなくなるまで繰り返す.
   (i) 集合 M の任意の要素を (s, P) とする．文形式 s の最左位置の非終端記号を N とし， N を左辺に持つ i 番目の規則で， s を書き換えた結果を t とする．$(t, P{\cdot}i)$ がまだ集合 M に含まれておらず，しかも $|t| \leq |x|$ なら，$(t, P{\cdot}i)$ を集合 M に追加する.
3. 集合 M の元で (x, P) の形の要素があれば，それぞれの P を左解析列として出力する．無ければエラーメッセージを出力する．　□

例 5.3 (単調増加法)

次の文法の例で入力 abba の場合を，方法 5.3 で構文解析してみると，

0) $S' \to S$	3) $A \to a$	6) $B \to b$
1) $S \to a\ B$	4) $A \to a\ S$	7) $B \to b\ S$
2) $S \to b\ A$	5) $A \to b\ A\ A$	8) $B \to a\ B\ B$

集合 M は，例えば次のように変化していく.

$$
\begin{aligned}
&\{(S, 0)\}\\
\Rightarrow\ &\{(aB, 01), (bA, 02)\}\\
\Rightarrow\ &\{(ab, 016), (abS, 017), (aaBB, 018), (ba, 023),\\
&(baS, 024), (bbAA, 025)\ \}\\
\Rightarrow\ &\{(ab, 016), (abaB, 0171), (abbA, 0172), (aabB, 0186), (ba, 023),\\
&(baaB, 0241), (babA, 0242), (bbaA, 0253)\}\\
\Rightarrow\ &\{(ab, 016), (abab, 01716), (abba, 01723), (aabb, 01866),\\
&(ba, 023), (baab, 02416), (baba, 02423), (bbaa, 02533)\}
\end{aligned}
$$

これ以上は集合の要素が追加できないので終了する．入力 abba の左解析列は，01723 と判明する． □

単調増加法は，右辺の長さ 0 の規則を含む文法の場合には，そのまま適用できないが，このアイデアは文脈依存文法の場合にも通用する．しかしながら，解析時間は最悪の場合，終端記号列の長さに対し指数関数的に増大するという弱点がある．

5.2 下降型構文解析

下降型構文解析の代表的方法である LL(k) 構文解析法について述べる．LL(k) 構文解析法は，LL(k) 構文解析表を用いて，スタック内で最左導出の再現を行いながら，入力列の長さに比例した時間で解析が終了する．

まず LL(1) 構文解析表を用いる LL(1) 構文解析法の例を，算術式の場合で示す．通常の算術式の文法は以下の形であるが，後でみるように左再帰の非終端記号が扱えないため，

1) $\mathrm{E} \to \mathrm{E} + \mathrm{T}$	4) $\mathrm{T} \to \mathrm{F}$
2) $\mathrm{E} \to \mathrm{T}$	5) $\mathrm{F} \to (\ \mathrm{E}\ )$
3) $\mathrm{T} \to \mathrm{T} * \mathrm{F}$	6) $\mathrm{F} \to \mathrm{i}$

右再帰の非終端記号を使う以下の文法を用いる．ちなみに左再帰の非終端記号を右再帰の非終端記号へ変換することはいつも可能である．

1) $\mathrm{E} \to \mathrm{T}\ \mathrm{E}'$	5) $\mathrm{T}' \to *\ \mathrm{F}\ \mathrm{T}'$
2) $\mathrm{E}' \to +\ \mathrm{T}\ \mathrm{E}'$	6) $\mathrm{T}' \to \varepsilon$
3) $\mathrm{E}' \to \varepsilon$	7) $\mathrm{F} \to (\ \mathrm{E}\ )$
4) $\mathrm{T} \to \mathrm{F}\ \mathrm{T}'$	8) $\mathrm{F} \to \mathrm{i}$

LL(k) 構文解析表は，文法記号 X またはスタックの底を示す記号 ⊥ と，長さ k 以下の先読み記号列 y に対して以下の 4 種類の記入項目を持つ．

“記号列 α, 生成規則番号 i” の形の記入項目は，展開を示す．つまりスタックの先頭記号 X を記号列 α に置き換え (すなわち，スタックの先頭記号 X をポップし，記号列 α の左右反転列を 1 記号ずつプッシュする)，解析列に

表 5.3 LL(1) 構文解析表

	i	+	*	(	)	$
E	TE′, 1			TE′, 1		
E′		+TE′, 2			ε, 3	ε, 3
T	FT′, 4			FT′, 4		
T′		ε, 6	*FT′, 5		ε, 6	ε, 6
F	i, 8			(E), 7		
i	pop					
+		pop				
*			pop			
(				pop		
)					pop	
⊥						accept

生成規則番号 i を出力することを示す．

“pop” の形の記入項目は，スタックの先頭記号 X をポップし，入力列の先頭 1 記号を消去することを示す．

“accept” の形の記入項目は，解析列に記号 a を出力し，正常終了することを示す．

“　” の形の記入項目は，エラーメッセージを出力し，異常終了することを示す．

上記の算術式の文法の場合の LL(1) 構文解析表は表 5.3 のようになる．LL(1) 構文解析表を用いる LL(1) 構文解析法を方法 5.4 に示す．

方法 5.4 (LL(1) 構文解析)

入力: LL(1) 構文解析表 M と入力列 x.

出力: 入力列 x の左解析列またはエラーメッセージ.

手順:

1. 開始時にスタックは記号列 S⊥ を保持し，長さ 1 の先読みは入力列の先頭 1 記号を指している．
2. 以下を永遠に繰り返す．スタックの先頭記号 X と入力列の先読み y の対に対し，定められた動作 M(X, y) に従い，(i) から (iv) の動作の 1 つ

表 **5.4** LL(1) 解析過程

入力列	スタック	解析列
i+i∗i$	E⊥	
i+i∗i$	TE′⊥	1
i+i∗i$	FT′E′⊥	14
i+i∗i$	iT′E′⊥	148
+i∗i$	T′E′⊥	148
+i∗i$	E′⊥	1486
+i∗i$	+TE′⊥	14862
i∗i$	TE′⊥	14862
i∗i$	FT′E′⊥	148624
i∗i$	iT′E′⊥	1486248
∗i$	T′E′⊥	1486248
∗i$	∗FT′E′⊥	14862486
i$	FT′E′⊥	14862486
i$	iT′E′⊥	148624868
$	T′E′⊥	148624868
$	E′⊥	1486248686
$	⊥	14862486863
$	⊥	14862486863a

を行う.

(i) **(展開)** 動作が展開なら，スタックの先頭記号を展開すべき記号列に置き換える．対応している規則番号を解析列に出力する．

(ii) **(照合)** スタックの先頭記号 1 記号と入力列の先頭記号 1 記号を消去する.

(iii) **(正常終了)** 解析列に記号 a を出力し，構文解析を正常終了する.

(iv) **(異常終了)** エラーメッセージを出力し，構文解析を異常終了する. □

例 5.4 (LL(1) 解析)

入力 i+i∗i に対する解析過程は，表 5.4 のようになる. □

$k \geq 1$ の一般の場合の LL(k) 構文解析法を次に示す.

方法 5.5 (LL(k) 構文解析)

入力: LL(k) 構文解析表と入力列 x.

出力: 入力列 x の左解析列またはエラーメッセージ.

手順:

1. 開始時にスタックは記号列 $[S,\{\$^k\}]\bot$ を保持し，長さ k の先読みは入力列の先頭 k 記号を指している．LL(k) 構文解析表は，スタックの先頭記号と入力列の先読みの対で動作が求まる.
2. 以下を永遠に繰り返す．スタックの先頭記号と入力列の先読みの対に対し，(i) から (iv) までのいずれか 1 つの定められた動作を行う.
   - (i) (**展開**) 動作が展開なら，スタックの先頭記号を展開すべき記号列に置き換える．対応している規則番号を解析列に出力する.
   - (ii) (**シフト**) スタックの先頭記号 1 記号と入力列の先頭記号 1 記号を消去する.
   - (iii) (**正常終了**) 解析列に記号 a を出力し，構文解析を正常終了する.
   - (iv) (**異常終了**) エラーメッセージを出力し，構文解析を異常終了する. □

LL(k) 構文解析表の構成法を 3 種類示す．ここでは，Wood 法，Lewis-Stearns 法，Tokuda 法による LL(k) 構文解析表の構成法を示す.

定義 5.3 (LL(k) 文法)

文法 G と非負整数 k の下で，次のような 2 つの最左導出があり，

$$S\$^k \overset{*}{\Longrightarrow} wA\gamma \Rightarrow w\alpha\gamma \overset{*}{\Longrightarrow} wx,$$
$$S\$^k \overset{*}{\Longrightarrow} wA\gamma \Rightarrow w\beta\gamma \overset{*}{\Longrightarrow} wy,$$

さらに，$\mathrm{FIRST}_k(x) = \mathrm{FIRST}_k(y)$ ならば，生成規則 $A \to \alpha$ と生成規則 $A \to \beta$ は同一であると必ず断定できる場合，文法 G を **LL(k) 文法** (LL(k) grammar) という. □

定義 5.4 (強 LL(k) 文法)

文法 G と非負整数 k の下で，次のような 2 つの最左導出があり，

$$S\$^k \overset{*}{\Longrightarrow} wA\gamma \Rightarrow w\alpha\gamma \overset{*}{\Longrightarrow} wx,$$

$$S\$^k \overset{*}{\Longrightarrow} w'A\gamma' \Rightarrow w'\beta\gamma' \overset{*}{\Longrightarrow} w'y,$$

さらに，$\mathrm{FIRST}_k(x) = \mathrm{FIRST}_k(y)$ ならば，生成規則 $A \to \alpha$ と生成規則 $A \to \beta$ は同一であると必ず断定できる場合，文法 G を**強 LL(*k*) 文法** (strong LL(*k*) grammar) という. □

$k > 1$ の場合，強 LL(*k*) 文法は LL(*k*) 文法の真の部分クラスである. $k = 1$ の場合， LL(1) 文法は強 LL(1) 文法と同一となる．強 LL(*k*) 構文解析表を作る Wood 法を示す.

方法 5.6 (Wood 法)

入力: 強 LL(*k*) 文法 G.

出力: 強 LL(*k*) 構文解析表 M.

手順:

1. 次のように構文解析表 M を記入する.
   (i) j 番目の規則 $A \to \alpha$ と，$\mathrm{FIRST}_k(\alpha) \oplus_k \mathrm{FOLLOW}_k(A)$ の要素である先読み列 u に対して，次のように定義する.
   $M(A, u) =$ "α, j".
   (ii) M(a, av) = "pop".
   ただし，v は全体の長さが $k-1$ の終端記号と右端記号列の結合列.
   (iii) $M(\perp, \$^k) =$ "accept".
   (iv) 上記以外の項目を "error" とする. □

例 5.5 (LL(2) 解析表)

次の文法は LL(2) 文法である.

1) $S \to a\ a$	3) $T \to (\ T\ )$
2) $S \to T$	4) $T \to (\ a\ )$

この文法用の強 LL(2) 構文解析表は表 5.5 のようになる．ただし，後の解析表と比較しやすいように，記号 X を記号 [X:*] で表している. □

LL(*k*) 構文解析表を作る Lewis-Stearns 法を示す.

表 5.5　強 LL(2) 構文解析表

	aa	((	(a
[S:*]	[a:*][a:*],1	[T:*],2	[T:*],2
[T:*]		[(:*][T:*][):*],3	[(:*][a:*][):*],4
[a:*]	pop		
[(:*]		pop	pop
[):*]			
[⊥:*]			

	a$	a)	)$	))	$$
[S:*]					
[T:*]					
[a:*]	pop	pop			
[(:*]					
[):*]			pop	pop	
[⊥:*]					accept

方法 5.7 (Lewis-Stearns 法)

入力: LL(k) 文法 G.

出力: LL(k) 構文解析表 M.

手順:

1. 状態の集合 J = {[S, {$\$^k$}]} とする.
2. ステップ 3. を新しい状態がつけ加えられなくなるまで行い，ステップ 4. へ行く.
3. J のそれぞれの状態 [A, L] について，左辺が A であるそれぞれの規則 (各 x_i は長さ 0 以上の終端記号列)

   $\mathrm{A} \to \mathrm{x}_0\ \mathrm{B}_1\ \mathrm{x}_1 \cdots \mathrm{B}_n\ \mathrm{x}_n$

   に対し，次のような局所的後続列の集合 Y_i $(0 \leq i \leq n)$

   $\mathrm{Y}_i = \mathrm{FIRST}_k(\mathrm{x}_i\ \mathrm{B}_{i+1}\ \mathrm{x}_{i+1} \cdots \mathrm{B}_n\ \mathrm{x}_n) \oplus_k \mathrm{L}$

   を求める．そして状態 $[\mathrm{B}_1, \mathrm{Y}_1]$, $[\mathrm{B}_2, \mathrm{Y}_2]$, $\cdots$, $[\mathrm{B}_n, \mathrm{Y}_n]$ のうち，J の中に既出でない状態を J に加える.
4. 次のように構文解析表 M を記入する.

(i) $[A, L]$ が J の要素で，$A \to x_0\ B_1\ x_1 \cdots B_n\ x_n$ が j 番目の規則で，先読み列 u が $\mathrm{FIRST}_k(x_0\ B_1\ x_1 \cdots B_n\ x_n) \oplus_k L$ のとき，次のように定義する．

$M([A, L], u) =$ “$x_0\ [B_1, Y_1]\ x_1 \cdots [B_n, Y_n]\ x_n,\ j$”

ただし，$n = 0$ のときには，次のように定義する．

$M([A, L], u) =$ “$x_0,\ j$”．

(ii) M(a, av) = “pop”.

ただし，v は長さ 0 以上 $k-1$ 以下の終端記号列．

(iii) $M([\bot, \{\$^k\}], \$^k) =$ “accept”.

(iv) 上記以外の項目を “error” とする． □

例 5.6 (Lewis-Stearns 法)

Wood 法の場合と同じ以下の文法

1) S → a a	3) T → (T)
2) S → T	4) T → (a)

について LL(2) 構文解析表を作ると，表 5.6 のようになる． □

最左誤り位置で誤りを検出する LL(k) 構文解析表を作る Tokuda 法を示す．

方法 5.8 (Tokuda 法)

入力: LL(k) 文法 G.

出力: LL(k) 構文解析表 M.

手順:

1. 状態の集合 $J = \{[S, \{\$^k\}]\}$ とする．
2. ステップ 3. を新しい状態がつけ加えられなくなるまで行い，ステップ 4. へ行く．
3. J のそれぞれの状態 [A, L] について，左辺が A であるそれぞれの規則 (各 a_i は ε か終端記号),

   $A \to a_0\ B_1\ a_1 \cdots B_n\ a_n$

   に対し，次のような局所的後続列の集合 Y_i $(0 \le i \le n)$

表 5.6　LL(2) 構文解析表

	aa	((	(a
[S:$$]	[a:*][a:*],1	[T:$$],2	[T:$$],2
[T:$$]		[(:*][T:)$][):*],3	[(:*][a:*][):*],4
[T:)$]		[(:*][T:))][):*],3	[(:*][a:*][):*],4
[T:))]		[(:*][T:))][):*],3	[(:*][a:*][):*],4
[a:*]	pop		
[(:*]		pop	pop
[):*]			
[⊥:*]			

	a$	a)	)$	))	$$
[S:$$]					
[T:$$]					
[T:)$]					
[T:))]					
[a:*]	pop	pop			
[(:*]					
[):*]			pop	pop	
[⊥:*]					accept

$Y_i = \mathrm{FIRST}_k(a_i\ B_{i+1}\ a_{i+1} \cdots B_n\ a_n) \oplus_k L$

を求める．そして，状態 $[a_0, Y_0]$，$[B_1, Y_1]$，$\cdots$，$[B_n, Y_n]$，$[a_n, Y_n]$ のうち，左のフィールドが ε でなく，かつ J の中に既出でない状態を J に加える．

4. 次のように構文解析表 M を記入する．
   (i) [A, L] が J の要素で，$A \rightarrow a_0\ B_1\ a_1 \cdots B_n\ a_n$ が j 番目の規則で，u が $\mathrm{FIRST}_k(a_i\ B_{i+1}\ a_{i+1} \cdots B_n\ a_n) \oplus_k L$ の要素のとき，次のように定義する．

   $M([A, L], u) =$ “$[a_0, Y_0][B_1, Y_1]\cdots[B_n, Y_n][a_n, Y_n], j$”，

   ただし，$[\varepsilon, Y] = \varepsilon$ とする．

   (ii) [a, L] が J の要素のとき，すべての L の要素 u に対して次のように定義する．

$\mathrm{M}([\mathrm{a}, \mathrm{L}], \mathrm{u}) = $ “pop”.

(iii) $\mathrm{M}([\perp, \{\$^k\}], \$^k) = $ “accept”.

(iv) 上記以外の項目を “error” とする．　□

例 5.7 (Tokuda 法)

Wood 法の場合と同じ以下の文法

1) S → a a	3) T → (T)
2) S → T	4) T → (a)

について LL(2) 構文解析表を作ると，表 5.7 のようになる．Lewis-Stearns 法では，LL(k) 文法の場合，誤り検出が $k-1$ 位置遅れることがあるが，この方法では，LL(k) 解析の範囲内で，誤り検出を最も早期に行う．　□

5.3　上昇型構文解析

上昇型構文解析の代表的方法である LR(k) 構文解析法について述べる．LR(k) 構文解析法では，LR(k) 構文解析表とスタックを用いて，入力列の長さに比例した時間で，入力列の構文解析を終了する．算術式の構文を定義する以下の文法の場合で説明する．

0) E′ → E	5) T → T / F
1) E → E + T	6) T → F
2) E → E − T	7) F → (E)
3) E → T	8) F → i
4) T → T ∗ F	

この文法に対する LR(1) 構文解析表は，表 5.8 のようになる．LR(k) 構文解析表は，動作表 ACTION と遷移表 GOTO からなる．動作表 ACTION は，状態番号 i と長さ k の先読み x に対し，記入項目 ACTION(i, x) を定義する．記入項目は次の 4 種類がある．

1. シフト動作

　s で表す．

表 5.7　LL(2) 構文解析表

	aa	((	(a
[S:$$]	[a:aa][a:a$],1	[T:$$],2	[T:$$],2
[T:$$]		[(:(([T:)$][):$$],3	[(:(a][a:a)][):)$],4
[T:)$]		[(:(([T:))][):))],3	[(:(a][a:a)][):))],4
[T:))]		[(:(([T:))][):))],3	[(:(a][a:a)][):))],4
[a:aa]	pop		
[a:a$]			
[a:a)]			
[(:((]		pop	
[(:(a]			pop
[):)$]			
[):))]			
[⊥:$$]			

	a$	a)	)$	))	$$
[S:$$]					
[T:$$]					
[T:)$]					
[T:))]					
[a:aa]					
[a:a$]	pop				
[a:a)]		pop			
[(:((]					
[(:(a]					
[):)$]			pop		
[):))]				pop	
[⊥:$$]					accept

表 **5.8** LR(1) 構文解析表

	i	)	(	/	*	−	+	$	E	T	F
0	s5		s4						1	2	3
1						s7	s6	a			
2		r3		s9	s8	r3	r3	r3			
3		r6		r6	r6	r6	r6	r6			
4	s5		s4						10	2	3
5		r8		r8	r8	r8	r8	r8			
6	s5		s4							11	3
7	s5		s4							12	3
8	s5		s4								13
9	s5		s4								14
10		s15				s7	s6				
11		r1		s9	s8	r1	r1	r1			
12		r2		s9	s8	r2	r2	r2			
13		r4		r4	r4	r4	r4	r4			
14		r5		r5	r5	r5	r5	r5			
15		r7		r7	r7	r7	r7	r7			

2. 還元動作

   j 番目の生成規則での還元を rj で表す.

3. 正常終了

   a で表す.

4. 異常終了

   エラー終了を空白で表す.

遷移表 GOTO は，状態番号 i と文法記号 X に対し，記入項目 GOTO(i, X) を定義する．記入項目 j は，遷移先の状態番号を表す.

LR(1) 構文解析表では，長さ 1 の先読み列 x は終端記号か右端記号 $ なので，1 枚の表で表すことができる．すなわち，還元の動作の項目の数字を除き，数字は GOTO 表の値を記入することにする．例えば， r3 は 3 番の規則による還元を示すが，数字のみの 3 や s4 はそれぞれ遷移先の状態番号が 3 と 4 であることを示す.

LR(k) 構文解析表を用いる LR(k) 構文解析法は，次の通りである.

方法 5.9 (状態のみをしまう LR(k) 解析)

入力: LR(k) 構文解析表と入力列 x.

出力: 入力列 x の右解析列またはエラーメッセージ.

手順:

1. 空のスタックに状態番号 0 をプッシュする．入力列の先頭 k 記号が現在の先読みである．スタックの先頭の状態番号と先読みにより動作が定まる．
2. 以下を永遠に繰り返す．構文解析表から現在の動作を求め，次の (i) から (iv) のいずれかの動作を行う．
   (i) **(シフト動作)** 入力列の先頭 1 記号を消去し，スタックに新しい状態番号をプッシュする．
   (ii) **(還元動作)** スタックの中の状態番号を先頭から還元右辺の長さ分だけポップする．次に現在のスタックの先頭記号と還元左辺により新しい状態番号を定め，スタックにプッシュする．還元規則番号を解析列に出力する．
   (iii) **(正常終了)** 解析列に記号 a を出力し，解析を正常終了する．
   (iv) **(異常終了)** エラーメッセージを出力し，解析を異常終了する．　□

例 5.8 (状態のみをしまう LR(k) 解析)

入力 i+i$ の場合の解析過程は，表 5.9 のようになる．　□

LR(k) 解析表の種類について説明する．

定義 5.5 (LR(k) 文法)

与えられた文法 G と非負整数 k について，次のような G の拡大文法 G′ の 2 つの最右導出が存在し，

$$\mathrm{S}'\$^k \overset{*}{\Longrightarrow} \gamma \mathrm{Ax} \Rightarrow \gamma\alpha \mathrm{x},$$
$$\mathrm{S}'\$^k \overset{*}{\Longrightarrow} \delta \mathrm{Bw} \Rightarrow \gamma\alpha \mathrm{y},$$

しかも，$\mathrm{FIRST}_k(\mathrm{x}) = \mathrm{FIRST}_k(\mathrm{y})$ ならば，必ず $\gamma\mathrm{Ay} = \delta\mathrm{Bw}$ である，つまり，導出

表 5.9 解析過程

スタック	入力列	解析列
0	i+i$	
05	+i$	
03	+i$	8
02	+i$	86
01	+i$	863
016	i$	863
0165	$	863
0163	$	8638
01611	$	86386
01	$	863861
01	$	863861a

$$\delta \mathrm{Bw} \Rightarrow \gamma\alpha \mathrm{y}$$

は，必ず導出

$$\gamma \mathrm{Ay} \Rightarrow \gamma\alpha \mathrm{y}$$

である場合，G を **LR(*k*) 文法** (LR(*k*) grammar) という． □

LR(*k*) 解析表には，Knuth 型，SLR(*k*) 型，LALR(*k*) 型，Anderson 型，バイパス型などのたくさんの種類がある．いくつかの例をみてみよう．

例 5.9 (Knuth 型と SLR 型と LALR 型)
次の文法に対する

0) $S' \to S$　　　2) $S \to c$
1) $S \to a\ S\ b$

Knuth 型 LR(1) 構文解析表は表 5.10 のようになる．

SLR 型と LALR 型の LR(1) 構文解析表はこの場合同一であり，表 5.11 のようになる．Knuth 型に比べて，一般に状態の数が少なくて済むという特徴がある． □

表 5.10　LR(1) 構文解析表

	a	b	c	$	S
0	s2		s3		1
1				a	
2	s20		s30		4
3				r2	
4		s5			
5				r1	
20	s20		s30		40
30		r2			
40		s50			
50		r1			

表 5.11　LR(1) 構文解析表

	a	b	c	$	S
0	s2		s3		1
1				a	
2	s2		s3		4
3		r2		r2	
4		s5			
5		r1		r1	

例 5.10 (Knuth 型とバイパス型)
次の文法に対する

0) $E' \to E$　　3) $T \to T\ y$
1) $E \to z$　　4) $T \to y$
2) $E \to T$

Knuth 型 LR(1) 構文解析表は表 5.12 のようになる．

バイパス型 LR(1) 構文解析表は表 5.13 のようになる．1 番と 4 番の規則の還元を行わずに，つまり還元動作をバイパスして正しい構文解析を行う．対応する意味動作が存在しないときに効率的な方法である．　□

表 5.12　LR(1) 構文解析表

	y	z	$	E	T
0	s4	s2		1	3
1			a		
2			r1		
3	r5		r2		
4	r4		r4		
5	r3		r3		

表 5.13　LR(1) 構文解析表

	y	z	$	E	T
0	s2	s1			2
1			a		
2	s3		a		
3	r3		r3		

表 5.14 LR(1) 構文解析表

	a	b	c	$	S
0	s2		s3		1
1				a	
2	s2		s3		4
3	r2	r2	r2	r2	
4		s5			
5	r1	r1	r1	r1	

表 5.15 LR(1) 構文解析表

	y	z	$	E	T
0	s2	s1			2
1			a		
2	s3		a		
3	r3	r3	r3		

例 5.11 (Anderson の変形)
LR(k) 構文解析表に嘘の還元動作を加えて解析表を小型化する方法が Anderson の変形である．例えば，表 5.11 と表 5.12 の解析表に Anderson の変形を行うと，それぞれ表 5.14，表 5.15 のようになる．Knuth 型，SLR 型，LALR 型，バイパス型では，構文解析表の正しさが保存される．これらの方法では，余分な還元の導入をシフトが見破ることが保証されている．□

LR(k) 解析表の構成法を，Knuth 型，SLR(k) 型，LALR(k) 型について示す．

定義 5.6 (LR(k) 項目)
x を文法 G の終端記号と右端記号の長さ k の列，$\mathrm{A} \to \alpha\beta$ を G の生成規則とするとき，$[\mathrm{A} \to \alpha \cdot \beta, \mathrm{x}]$ という形の項目を LR(k) 項目という．文法 G の LR(k) 項目の集合 $\mathrm{ITEM}_k(\mathrm{G})$ を，次のように定義する．

$$\mathrm{ITEM}_k(\mathrm{G}) = \left\{ [\mathrm{A} \to \alpha \cdot \beta, \mathrm{x}] \;\middle|\; \begin{array}{l} \mathrm{A} \to \alpha\beta \text{ は G の生成規則，x は G} \\ \text{の終端記号と右端記号の長さ } k \text{ の列} \end{array} \right\}.$$

□

定義 5.7 (ふくらまし操作)
文法 G の LR(k) 項目の集合 $\mathrm{ITEM}_k(\mathrm{G})$ の上の関係 ↓ を，次のように定義する．

$$\mathrm{I}_1 \downarrow \mathrm{I}_2 \overset{\text{定義}}{\Longleftrightarrow} \mathrm{I}_1 = [\mathrm{A} \to \alpha{\cdot}\mathrm{B}\beta, \mathrm{x}],\ \mathrm{I}_2 = [\mathrm{B} \to {\cdot}\gamma, \mathrm{y}], \quad \mathrm{y} \text{ は } \mathrm{FIRST}_k(\beta\mathrm{x}).$$

$\mathrm{ITEM}_k(\mathrm{G})$ の部分集合 J に対し，

$$\mathrm{J}{\downarrow}* = \left\{ \mathrm{I}_n \,\middle|\, \begin{array}{l} \mathrm{I}_0 \text{ は J の要素，} n \text{ は 0 以上の整数，} \mathrm{I}_0 \downarrow \mathrm{I}_1, \cdots, \\ \mathrm{I}_{n-1} \downarrow \mathrm{I}_n \end{array} \right\}.$$

□

定義 5.8 (終端記号による引っ越し)

文法 G の LR(k) 項目の集合 $\mathrm{ITEM}_k(\mathrm{G})$ の上の関係 $\xrightarrow{\mathrm{X}}$ を，次のように定義する．ただし，X は文法記号とする．

$$\mathrm{I}_1 \xrightarrow{\mathrm{X}} \mathrm{I}_2 \overset{\text{定義}}{\Longleftrightarrow} \mathrm{I}_1 = [\mathrm{A} \to \alpha \cdot \mathrm{X}\,\beta, \mathrm{y}]\ ,\ \mathrm{I}_2 = [\mathrm{A} \to \alpha\, \mathrm{X} \cdot \beta, \mathrm{y}].$$

$\mathrm{ITEM}_k(\mathrm{G})$ の部分集合 J に対し，

$$\mathrm{J} \xrightarrow{\mathrm{X}} = \left\{ \mathrm{I}_2 \,\middle|\, \mathrm{I}_1 \text{ は J の要素，} \mathrm{I}_1 \xrightarrow{\mathrm{X}} \mathrm{I}_2 \right\}.$$

□

定義 5.9 (引っ越しする記号)

$\mathrm{ITEM}_k(\mathrm{G})$ の部分集合 J の SUCC(J) を，次のように定義する．

$$\mathrm{SUCC(J)} = \{\mathrm{X} \mid [\mathrm{A} \to \alpha \cdot \mathrm{X}\,\beta, \mathrm{y}] \text{ は J の要素}\}.$$

□

定義 5.10 (EFF_kの計算法)

$$\mathrm{EFF}_k(\alpha) = \left\{ \mathrm{z} \,\middle|\, \begin{array}{l} \mathrm{z} \text{ は } \mathrm{FIRST}_k(\alpha) \text{ の要素であり，しかも次のよう} \\ \text{な最右導出が存在する．} \\ \quad \alpha \overset{*}{\Longrightarrow} \mathrm{wAy} \Rightarrow \mathrm{wxy}, \ (\mathrm{w} \neq \varepsilon) \text{ または } (\mathrm{w} = \varepsilon \\ \text{で，} \mathrm{A} \to \mathrm{x} \text{ は非空規則)，} \mathrm{z} \text{ は } \mathrm{FIRST}_k(\mathrm{wxy}) \end{array} \right\}.$$

□

方法 5.10 (Knuth 法)

入力: 文脈自由文法 G.

出力: G が LR(k) 文法のとき，G に対する Knuth 型 LR(k) 解析表.

手順:

1. LR(k) 項目の集合 (状態と呼ぶ) の集合を C で表し，C = ϕ とする．次の初期状態 I_0 を “未処理” とマーク付けして，C に加える．

$I_0 = \{[S' \to \cdot\, S, \$^k]\}\downarrow *$.

2. C 内の未処理の状態を 1 つ選び，I と呼ぶ．状態 I を “既処理” とマーク付けする．
3. 状態 I の後続記号の集合 SUCC(I) を $\{X_1, X_2, \cdots, X_n\}$ $(n \geq 0)$ とする．i を 1 から n まで繰り返す．
   (i) $(I \xrightarrow{X_i})\downarrow *$ が C に既出の I′ と一致するのならば，$GOTO(I, X_i) = I'$ とする．
   (ii) $(I \xrightarrow{X_i})\downarrow *$ が C に既出の状態と一致しないのならば，新しい状態名 J と名づけ，$GOTO(I, X_i) = J$ とし，状態 J を “未処理” とマーク付けして，C に加える．
4. C 内の未処理の状態を 1 つ選び，I と呼ぶ．状態 I を “既処理” とマーク付けし，ステップ 3. へ行く．C 内に未処理の状態がなければ，ステップ 5. へ行く．
5. 動作を次のように定める．
   (i) $[A \to \alpha{\cdot}X\beta, w]$ が C の状態 I にあり，x が $EFF_k(X\beta w)$ の要素なら，ACTION(I, x) = “shift”.
   (ii) $[A \to \gamma\cdot, x]$ が C の状態 I にあり，$A \to \gamma$ が p 番目 $(p \neq 0)$ の規則ならば，ACTION(I, x) = “reduce p”.
   (iii) $[A \to \gamma\cdot, x]$ が C の状態 I にあり，$A \to \gamma$ が 0 番目の規則ならば，ACTION(I, x) = “ accept”.
   (iv) その他について，ACTION(I, x) = “error”. □

例 5.12 (Knuth 法)

次の文法に対する Knuth 型 LR(1) 解析表を作るために，

0) $E' \to E$　　3) $T \to T\ y$

1) $E \to z$　　4) $T \to y$

2) $E \to T$

LR(1) 項目集合の集合を数え上げると図 5.3 のようになる．表 5.16 のような LR(1) 項目集合と状態番号の対応を定めると，表 5.17 のような Knuth 型 LR(1) 構文解析表が得られる． □

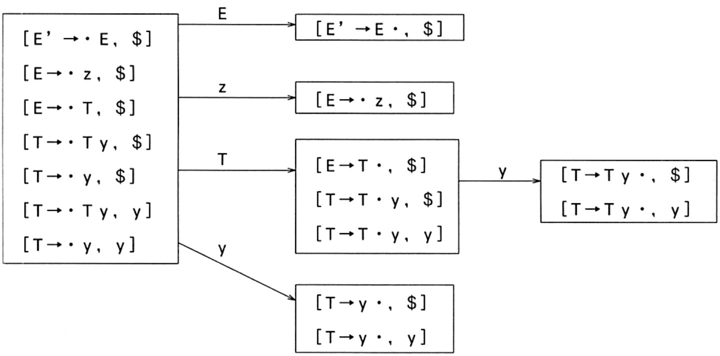


図 5.3　LR(1) 項目集合の数え上げ

表 5.16　LR(1) 項目集合の集合

状態 0	状態 2
[E′ → · E, $]	[E → z ·, $]
[E → · z, $]	状態 3
[E → · T, $]	[E → T ·, $]
[T → · T y, $]	[T → T ·y, $]
[T → · y, $]	[T → T ·y, y]
[T → · T y, y]	状態 4
[T → · y, y]	[T → y ·, $]
状態 1	[T → y·, y]
[E′ → E ·, $]	状態 5
	[T → T y ·, $]
	[T → T y ·, y]

表 5.17　LR(1) 構文解析表

	y	z	$	E	T
0	s4	s2		1	3
1			a		
2			r1		
3	s5		r2		
4	r4		r4		
5	r3		r3		

方法 5.11 (SLR 法)

入力: 文脈自由文法 G.

出力: 動作表に動作競合がなければ，G に対する正しい SLR(k) 解析表.

手順:

1. LR(0) 項目の集合の集合を C で表し，C $= \phi$ とする．次の初期状態 I_0 を "未処理" とマーク付けして，C に加える．

   $\mathrm{I}_0 = \{[\mathrm{S}' \to \cdot\ \mathrm{S}, \varepsilon]\}\downarrow *$.

2. C 内の未処理の状態を 1 つ選び，I と呼ぶ．状態 I を "既処理" とマーク付けする．

3. 状態 I の後続記号の集合 SUCC(I) を $\{\mathrm{X}_1, \mathrm{X}_2, \cdots, \mathrm{X}_n\}$ $(n \geq 0)$ とする．i を 1 から n まで繰り返す．

   (i) $(\mathrm{I} \xrightarrow{\mathrm{X}_i})\downarrow *$ が C に既出の I′ と一致するのならば，$\mathrm{GOTO}(\mathrm{I}, \mathrm{X}_i) = \mathrm{I}'$ とする．

   (ii) $(\mathrm{I} \xrightarrow{\mathrm{X}_i})\downarrow *$ が C に既出の状態と一致しないのならば，新しい状態名 J と名づけ，$\mathrm{GOTO}(\mathrm{I}, \mathrm{X}_i) = \mathrm{J}$ とし，状態 J を "未処理" とマーク付けして，C に加える．

4. C 内の未処理の状態を 1 つ選び，I と呼ぶ．状態 I を "既処理" とマーク付けし，ステップ 3. へ行く．C 内に未処理の状態がなければ，ステップ 5. へ行く．

5. 動作を次のように定める．

   (i) $[\mathrm{A} \to \alpha \cdot \mathrm{X}\ \beta, \varepsilon]$ が C の状態 I にあり，w が $\mathrm{FOLLOW}_k(\mathrm{A})$ に属し，x が $\mathrm{EFF}_k(\mathrm{X}\beta\mathrm{w})$ に属すなら，ACTION(I, x) = "shift".

   (ii) $[\mathrm{A} \to \gamma\ \cdot, \varepsilon]$ が C の状態 I にあり，$\mathrm{A} \to \gamma$ が p 番目 $(p \neq 0)$ の規則で，x が $\mathrm{FOLLOW}_k(\mathrm{A})$ に属すなら，ACTION(I, x) = "reduce p".

   (iii) $[\mathrm{A} \to \gamma\ \cdot, \varepsilon]$ が C の状態 I にあり，$\mathrm{A} \to \gamma$ が 0 番目の規則で，x が $\mathrm{FOLLOW}_k(\mathrm{A})$ に属すなら，ACTION(I, x) = "accept".

   (iv) その他について，ACTION(I, x) = "error".　□

例 5.13 (SLR 法)

まず，次の文法に対する Knuth 型 LR(1) 構文解析表を作る．

表 5.18　LR(1) 項目集合の集合

状態 0	状態 5
[S′ → · S, $]	[S → a S b ·, $]
[S → · a S b, $]	状態 20
[S → · c, $]	[S → a · S b, b]
状態 1	[S → · a S b, b]
[S′ → S ·, $]	[S → · c, b]
状態 2	状態 30
[S → a · S b, $]	[S → c ·, b]
[S → · a S b, b]	状態 40
[S → · c, b]	[S → a S · b, b]
状態 3	状態 50
[S → c ·, $]	[S → a S b ·, b]
状態 4	
[S → a S · b, $]	

表 5.19　LR(1) 構文解析表

	a	b	c	$	S
0	s2		s3		1
1				a	
2	s20		s30		4
3				r2	
4		s5			
5				r1	
20	s20		s30		40
30		r2			
40		s50			
50		r1			

0) S′ → S　　　　2) S → c
1) S → a S b

表 5.18 のような LR(1) 項目集合の集合と，表 5.19 のような Knuth 型 LR(1) 構文解析表が得られる．

同じ文法に対する SLR 型 LR(1) 構文解析表を作る．

0) S′ → S　　　　2) S → c
1) S → a S b

LR(0) 項目 [A → $\alpha \cdot \beta$, ε] の先読み列の所に，FOLLOW_1(A) の情報を利用して求めた先読み列を付けて表示すると，表 5.20 のような LR(1) 項目集合の集合となる．記法 [A → $\alpha \cdot \beta$, x/y] は [A → $\alpha \cdot \beta$, x] と [A → $\alpha \cdot \beta$, y] を示す．そして最終的に，表 5.21 のような SLR 型 LR(1) 構文解析表が得られる．解析動作の競合が発生しない限りは，Knuth 型の構文解析表より SLR 型の構文解析表を使用するほうが状態数が大きくならずに済む．　□

表 5.20 LR(1) 項目集合の集合

状態 0	状態 3
[S′ → · S, $]	[S → c ·, $]
[S → · a S b, $/b]	[S → c ·, b]
[S → · c, $/b]	状態 4
状態 1	[S → a S · b, $]
[S′ → S ·, $]	[S → a S · b, b]
状態 2	状態 5
[S → a · S b, $]	[S → a S b ·, $]
[S → a · S b, b]	[S → a S b ·, b]
[S → · a S b, $/b]	
[S → · c, $/b]	

表 5.21 LR(1) 構文解析表

	a	b	c	$	S
0	s2		s3		1
1				a	
2	s2		s3		4
3		r2		r2	
4		s5			
5		r1		r1	

方法 5.12 (LALR 法)

入力: 文脈自由文法 G.

出力: 動作表に動作競合がなければ，G に対する正しい LALR(k) 構文解析表.

手順:

1. LR(k) 項目の集合の集合を C で表し，$C = \phi$ とする．次の初期状態 I_0 を “未処理” とマーク付けして，C に加える．

   $I_0 = \{[S' \to \cdot\ S,\ \$^k]\}{\downarrow}^*$.

2. C 内の未処理の状態を 1 つ選び，I と呼ぶ．状態 I を “既処理” とマーク付けする．
3. 状態 I の後続記号の集合 SUCC(I) を $\{X_1, X_2, \cdots, X_n\}$ $(n \geq 0)$ とする．i を 1 から n まで繰り返す．

   (i) $(I \xrightarrow{X_i})\downarrow^*$ が C に既出の I′ と一致するのならば，$\mathrm{GOTO}(I, X_i) = I'$ とする．

   (ii) $(I \xrightarrow{X_i})\downarrow^*$ が C に既出の状態と一致しないのならば，新しい状態名 J と名づけ，$\mathrm{GOTO}(I, X_i) = J$ とし，状態 J を “未処理” とマーク付けして，C に加える．

4. C 内の未処理の状態を 1 つ選び，I と呼ぶ．状態 I を “既処理” とマーク付けし，ステップ 3. へ行く．C 内に未処理の状態がなければ，ステッ

プ 5. へ行く.

5. C を，$C = \{I_1, I_2, \cdots, I_m\}$ とする．C の状態の上に，次の同値関係 $\sim$ を定義する.

$$I \sim J \overset{定義}{\iff} \mathrm{CORE}(I) = \mathrm{CORE}(J).$$

ただし，$\mathrm{CORE}(I) = \{[A \to \alpha \cdot \beta] \mid [A \to \alpha \cdot \beta, w]$ は I の要素$\}$．C の $\sim$ による同値類を $\ll C_1 \gg, \ll C_2 \gg, \cdots, \ll C_l \gg$ とし，

$$J_i = \{t \mid 状態 I は \ll C_i \gg の要素，項目 t は状態 I の要素\}$$

として LR(k) 項目の集合の集合 $D = \{J_1, J_2, \cdots, J_l\}$ を作る.

6. 動作を次のように定める.
   (i) $[A \to \alpha \cdot X\beta, w]$ が D の状態 J にあり，x が $\mathrm{EFF}_k(X\beta w)$ の要素なら，ACTION(J, x) = “shift”.
   (ii) $[A \to \gamma \cdot, x]$ が D の状態 J にあり，$A \to \gamma$ が p 番目 ($p \neq 0$) の規則ならば，ACTION(J, x) = “reduce p”.
   (iii) $[A \to \gamma \cdot, x]$ が D の状態 I にあり，$A \to \gamma$ が 0 番目の規則ならば，ACTION(J, x) = “accept”.
   (iv) その他について，ACTION(J, x) = “error”.　□

例 5.14 (LALR 法)

まず，次の文法に対する Knuth 型 LR(1) 構文解析表を作ると，

0) $S' \to S$	3) $L \to * R$
1) $S \to L = R$	4) $L \to i$
2) $S \to R$	5) $R \to L$

表 5.22 のような LR(1) 項目集合の集合と，表 5.23 のような Knuth 型 LR(1) 構文解析表が得られる.

同じ文法に対する LALR 型 LR(1) 構文解析表を作ると，表 5.24 のような LR(1) 項目集合の集合と，表 5.25 のような LALR 型 LR(1) 解析表が得られる．LALR 型の解析表は，SLR 型でしばしば発生する動作競合が起こりにくくなるように改良したものである.　□

次にバイパス型の LR(k) 構文解析表の構成法を示す．$X \to Y$ の形をした生成規則を**単位規則** (unit production) という．G の 0 番目の規則を除

表 5.22 LR(1) 項目集合の集合

状態 0:	状態 3:	状態 6:	状態 80:
[S′ → · S, $]	[S → R ·, $]	[S → L = · R, $]	[R → L ·, $]
[S → · L = R, $]	状態 4:	[R → · L, $]	状態 40:
[S → · R, $]	[L → ∗ · R, =]	[L → · ∗ R, $]	[L → ∗ · R, $]
[L → · ∗ R, =]	[L → ∗ · R, $]	[L → · i, $]	[R → · L, $]
[L → · i, =]	[R → · L, =]	状態 7:	[L → · ∗ R, $]
[R → · L, $]	[L → · ∗ R, =]	[L → ∗ R ·, =]	[L → · i, $]
[L → · ∗ R, $]	[L → · i, =]	[L → ∗ R ·, $]	状態 50:
[L → · i, $]	[R → · L, $]	状態 8:	[L → i ·, $]
状態 1:	[L → · ∗ R, $]	[R → L ·, =]	状態 70:
[S′ → S ·, $]	[L → · i, $]	[R → L ·, $]	[L → ∗ R ·, $]
状態 2:	状態 5:	状態 9:	
[S → L · = R, $]	[L → i ·, =]	[S → L = R ·, $]	
[R → L ·, $]	[L → i ·, $]		

表 5.23 LR(1) 構文解析表

	∗	=	i	$	S	L	R
0	s4		s5		1	2	3
1				a			
2		s6		r5			
3				r2			
4	s4		s5			8	7
5		r4		r4			
6	s40		s50				
7		r3		r3			
8		r5		r5			
9				r1			
80				r5			
40	s40		s50			80	70
50				r4			
70				r3			

表 **5.24**　LR(1) 項目集合の集合

状態 0:	状態 3:	状態 6:
[S′ → · S, $]	[S → R ·, $]	[S → L = · R, $]
[S → ·L = R, $]	状態 4:	[R → · L, $]
[S → · R, $]	[L → * · R, =]	[L → · * R, $]
[L → · * R, =]	[L → * · R, $]	[L → · i, $]
[L → · i, =]	[R → · L, =]	状態 7:
[R → · L, $]	[L → · * R, =]	[L → * R ·, =]
[L → · * R, $]	[L → · i, =]	[L → * R ·, $]
[L → · i, $]	[R → · L, $]	状態 8:
状態 1:	[L → · * R, $]	[R → L ·, =]
[S′ → S ·, $]	[L → · i, $]	[R → L ·, $]
状態 2:	状態 5:	状態 9:
[S → L · = R, $]	[L → i ·, =]	[S → L=R·, $]
[R → L ·, $]	[L → i ·, $]	

表 **5.25**　LR(1) 構文解析表

	*	=	i	$	S	L	R
0	s4		s5		1	2	3
1				a			
2		s6		r5			
3				r2			
4	s4		s5			8	7
5		r4		r4			
6	s4		s5				
7		r3		r3			
8		r5		r5			
9				r1			

く，0 個以上の単位規則の集合を U で表す．これをバイパス規則の集合という．すべての A 規則が U の要素なら，非終端記号 A を黒い非終端記号という．どの A 規則も U の要素でないなら，非終端記号 A を白い非終端記号という．ある A 規則は U の要素で，別の A 規則は U の要素でないなら，非終端記号 A を灰色の非終端記号という．

定義 5.11 (関係 $\triangledown$)
文法 G の文法記号の上の関係 $\triangledown$ を，次のように定義する．

X $\triangledown$ Y とは，X $\rightarrow$ Y が U の規則

X $\triangledown*$ Y とは，X と Y が同一の記号か，X $\triangledown*$ Z，Z $\triangledown$ Y となる Z が存在すること． □

定義 5.12 (LR(k) 項目)
文法 G のバイパス規則の集合 U に関する LR(k) 項目の集合 $\mathrm{ITEM}_k[\mathrm{U}](\mathrm{G})$ を，次のように定義する．

$$\mathrm{ITEM}_k[\mathrm{U}](\mathrm{G}) = \left\{ [\mathrm{A} \rightarrow \alpha \cdot \beta,\ \mathrm{x}] \;\middle|\; \begin{array}{l} \mathrm{A} \rightarrow \alpha\beta \text{ は U の要素でない G} \\ \text{の生成規則，x は G の終端記} \\ \text{号と右端記号の長さ } k \text{ の列} \end{array} \right\}.$$

□

定義 5.13 (ふくらまし操作)
文法 G のバイパス規則の集合 U に関する LR(k) 項目の集合 $\mathrm{ITEM}_k[\mathrm{U}](\mathrm{G})$ の上の関係 $\downarrow$ を，次のように定義する．

$\mathrm{I}_1 \downarrow \mathrm{I}_2$ とは，

$\mathrm{I}_1 = [\mathrm{A} \rightarrow \alpha{\cdot}\mathrm{B}\beta,\ \mathrm{x}]$，B $\triangledown*$ C，$\mathrm{I}_2 = [\mathrm{C} \rightarrow {\cdot}\gamma,\ \mathrm{y}]$，y は $\mathrm{FIRST}_k(\beta\mathrm{x})$ に属すこと．

$\mathrm{ITEM}_k[\mathrm{U}](\mathrm{G})$ の部分集合 J に対し，

$$\mathrm{J}\downarrow * = \left\{ \mathrm{I}_n \;\middle|\; \begin{array}{l} \mathrm{I}_0 \text{ は J の要素，} n \text{ は 0 以上の整数, } \mathrm{I}_0 \downarrow \mathrm{I}_1, \\ \cdots, \mathrm{I}_{n-1} \downarrow \mathrm{I}_n \end{array} \right\}.$$ □

定義 5.14 (引っ越し操作)
文法 G のバイパス規則の集合 U に関する LR(k) 項目の集合 $\mathrm{ITEM}_k[\mathrm{U}](\mathrm{G})$ の上の関係 $\xrightarrow{\mathrm{X}}$ を，次のように定義する．ただし， X は文法記号とする．

$\mathrm{I}_1 \xrightarrow{\mathrm{X}} \mathrm{I}_2$ とは,

$\mathrm{I}_1 = [\mathrm{A} \to \alpha\cdot\mathrm{X}\beta, \mathrm{y}]$ で，$\mathrm{I}_2 = [\mathrm{A} \to \alpha\mathrm{X}\cdot\beta, \mathrm{y}]$ のこと．

$\mathrm{ITEM}_k[\mathrm{U}](\mathrm{G})$ の部分集合 J に対し,

$\mathrm{J} \xrightarrow{\mathrm{X}} = \{\mathrm{I}_2 \mid \mathrm{I}_1$ は J の要素，$\mathrm{I}_1 \xrightarrow{\mathrm{X}} \mathrm{I}_2\}$. □

定義 5.15 (引っ越し記号)
$\mathrm{ITEM}_k[\mathrm{U}](\mathrm{G})$ の部分集合 J の後続記号の列 $\mathrm{SUCC(J)} = (\mathrm{X}_1, \mathrm{X}_2, \cdots, \mathrm{X}_n)$ を，次のように定義する．

1. $\{\mathrm{X}_1, \mathrm{X}_2, \cdots, \mathrm{X}_n\} = \{\mathrm{Y} \mid [\mathrm{A} \to \alpha\cdot\mathrm{X}\beta, \mathrm{y}]$ は J の要素，$\mathrm{X} \bigtriangledown\!* \mathrm{Y}\}$
2. $i > j$ のとき，$\mathrm{X}_i \bigtriangledown\!* \mathrm{X}_j$ となることはない． □

方法 5.13 (バイパス法)
入力: LR(k) 文法 G と G の単位規則の集合 U.
出力: バイパス型 LR(k) 構文解析表.
手順:

1. $\mathrm{ITEM}_k[\mathrm{U}](\mathrm{G})$(状態と呼ぶ) の集合を C で表し，$\mathrm{C} = \phi$ とする．次の初期状態 I_0 を "未処理" とマーク付けして，C に加える．
   $\mathrm{I}_0 = \{[\mathrm{S}' \to \cdot\ \mathrm{S}, \$^k]\}\!\downarrow *$.
2. C 内の未処理の状態を 1 つ選び，I と呼ぶ．状態 I を "既処理" とマーク付けする．
3. 状態 I の後続記号の列 SUCC(I) を $(\mathrm{X}_1, \mathrm{X}_2, \cdots, \mathrm{X}_n)(n \geq 0)$ とする．$\mathrm{T}_1 = \mathrm{T}_2 = \cdots = \mathrm{T}_n = \phi$ とする．i を 1 から n まで繰り返す．
   (i) $(\mathrm{I} \xrightarrow{\mathrm{X}_i})\downarrow *$ を T_i に加える．
      a. X_i が終端記号か白い非終端記号の場合．
      T_i が C に既出の I′ と一致するのならば，$\mathrm{GOTO}(\mathrm{I}, \mathrm{X}_i) = \mathrm{I}'$ とする．T_i が C に既出の状態と一致しないのならば，新しい状態

名 J と名づけ，GOTO(I, X_i) = J とし，状態 J を “未処理” とマーク付けして，C に加える．

b. 黒い非終端記号の場合．

T_i の内容を，$X_i \triangledown X_j$ となるすべての T_j へコピーする．$T_i = \phi$ とする．

c. X_i が灰色の非終端記号の場合．

T_i の内容を，$X_i \triangledown X_j$ となるすべての T_j へコピーする．T_i が C に既出の I′ と一致するのならば，GOTO(I, X_i) = I′ とする．T_i が C に既出の状態と一致しないのならば，新しい状態名 J と名づけ，GOTO(I, X_i) = J とし，状態 J を “未処理” とマーク付けして，C に加える．

(ii) C 内の未処理の状態を 1 つ選び，I と呼ぶ．状態 I を “既処理” とマーク付けし，ステップ 3. へ．C 内に未処理の状態がなければ，ステップ 5. へ．

(iii) 動作を次のように定める．

a. [A → $\alpha \cdot X_i \beta$, w] が C の状態 I にあり，x が $EFF_k(X_i \beta w)$ に属すなら，ACTION(I, x) = “shift”．

b. [A → $\gamma \cdot$, x] が C の状態 I にあり，A → γ が p 番目 ($p \neq 0$) の規則ならば，ACTION(I, x) = “reduce p”．

c. [A → $\gamma \cdot$, x] が C の状態 I にあり，A → γ が 0 番目の規則ならば，ACTION(I, x) = “ accept”．

d. その他について，ACTION(I, x) = “error”． □

例 5.15 (バイパス法)

次の文法に対し，

0) E′ → E	3) T → T y
1) E → z	4) T → y
2) E → T	

バイパスする単位規則として

表 5.26　LR(1) 項目集合の集合

状態 0	状態 2
$[E' \to \cdot\ E,\ \$]$	$[E' \to E\ \cdot,\ \$]$
$[T \to \cdot\ T\ y,\ \$]$	$[T \to T\ \cdot\ y,\ \$]$
$[T \to \cdot\ T\ y,\ y]$	$[T \to T\ \cdot\ y,\ y]$
状態 1	状態 3
$[E' \to E\ \cdot,\ \$]$	$[T \to T\ y\ \cdot,\ \$]$
	$[T \to T\ y\ \cdot,\ y]$

表 5.27　LR(1) 構文解析表

	y	z	$	E	T
0	s2	s1			2
1			a		
2	s3		a		
3	r3		r3		

1) $E \to z$　　4) $T \to y$

2) $E \to T$

を選んで，バイパス型 LR(1) 構文解析表を作ると，表 5.26 のような LR(1) 項目集合の集合と，表 5.27 のようなバイパス型 LR(1) 解析表が得られる．

□

実際のプログラミング言語の構文規則の記述には，LR(0) 文法や SLR(1) 文法では制限が強過ぎるが，LALR(1) 文法であれば，ほぼ記述できることが知られている．ただ，正しい構文規則という概念を厳格に考えると，**6.4** で示すように，文脈自由文法では完全には記述できないことが知られている．

5.4　混合型構文解析

混合型構文解析の代表的方法である LC(k) 構文解析について述べる．LC(k) 構文解析では以下のような構文解析木 T の LC たどりを用いて，生成規則を認識する．LC たどりは，木 T の第 1 部分木を下から上へ，第 2 部分木から最右部分木までをそれぞれ上から下へとたどるので，下降型と上昇型の混合型とみなすことができる．LC たどりの概念図を図 5.4 に示す．

LC たどり (T):

1. T が葉のみの場合，
   訪問 (根 (T))
2. T が葉のみでない場合，

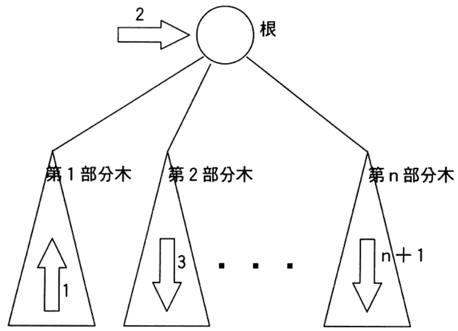


図 5.4 LC たどりの概念図

LC たどり (第 1 部分木 (I));
訪問 (根 (T));
出力 (生成規則番号);
LC たどり (第 2 部分木 (I));
...
LC たどり (最右部分木 (I))

LC(k) 構文解析では LC(k) 構文解析表を用いて構文解析を行う．LC(k) 構文解析表は次のような 5 種類の記入項目で解析動作を指定する．記入項目は，スタックの先頭の記号と長さ k 以下の先読み記号列 x に対して定義する．スタックの先頭の記号は，非終端記号 A と生成規則 B $\rightarrow \alpha$ の組 [A, B $\rightarrow \alpha$]，または非終端記号 A のいずれかである．構文解析の開始状態では，スタックは開始記号のみを保持している．

1. T(A, x) = (α[A, B $\rightarrow \alpha$], i)
   この動作指定は，スタックの先頭記号が A で先読みが x の場合，スタックの先頭記号 A を記号列 α[A, B $\rightarrow \alpha$] に置き換え，生成規則番号 i を出力することを意味する．
2. T([A, A $\rightarrow \tau$], x) = (ε, ε)

この動作指定は，スタックの先頭記号が [A, A $\rightarrow \tau$] で先読みが x の場合，スタックの先頭記号 [A, A $\rightarrow \tau$] を空列に置き換え，何も生成規則番号は出力しないことを意味する．

3. T(a, ay) = “pop”

この動作指定は，スタックの先頭記号が a で先読みが ay の場合，スタックの先頭記号 a を空列に置き換えることを意味する．

4. T($, ε) = “accept”

この動作指定は，スタックの先頭記号が $ で先読みが ε の場合，解析列に記号 a を出力して，解析を正常終了することを意味する．

5. “error”

この動作指定は，エラーメッセージを出力し，構文解析を異常終了することを意味する．“error” は空白で表す． □

例 5.16 (LC(1) 構文解析表)

左再帰の非終端記号 L を持つ以下の文法

1) S → . L　　　4) B → 1
2) L → L B　　　5) B → 0
3) L → B

に対する LC(1) 構文解析表を作ると，表 5.28 のようになる．非終端記号 A と生成規則 B $\rightarrow \alpha$ の組 [A, B $\rightarrow \alpha$] は，LC 項目と呼ばれる．LC 項目は動作をわかりやすくするために，[A, B $\rightarrow \alpha$] と冗長に書いている．構文解析のためだけであれば，生成規則がどの B 規則であるかを区別せず，[A, B] と書くだけでよい． □

定義 5.16 (左隅出現でない非終端記号)

左文形式 wAδ の最左位置の非終端記号 A は，生成規則の左隅の記号として，導入されたものではないとき，左隅出現でない非終端記号 A という．左隅出現でない左文形式 wAδ の非終端記号 A を，次のように

$$S \overset{*}{\underset{lc}{\Longrightarrow}} wA\delta$$

表 5.28　LC(1) 構文解析表

	1	0	.	ε
S			.L[S, S → .L], 1	
L	1[L, B → 1], 4	0[L, B → 0], 5		
B	1[B, B → 1], 4	0[B, B → 0], 5		
[S, S → .L]				ε, ε
[L, B → 1]	[L, L → B], 3	[L, L → B], 3		[L, L → B], 3
[L, B → 0]	[L, L → B], 3	[L, L → B], 3		[L, L → B], 3
[L, L → LB]	B[L, L → LB], 2	B[L, L → LB], 2		ε, ε
[L, L → B]	B[L, L → LB], 2	B[L, L → LB], 2		ε, ε
[B, B → 1]	ε, ε	ε, ε		ε, ε
[B, B → 0]	ε, ε	ε, ε		ε, ε
1	pop			
0		pop		
.			pop	
$				accept

書くことにする．なお，開始記号 S は左文形式 S において左隅出現でない非終端記号とする． □

定義 5.17 (LC(k) 文法)

以下の条件を満たす文脈自由文法 G を，**LC(k)** 文法という．次のような左隅出現でない任意の非終端記号 A

$$S \overset{*}{\underset{\mathrm{lc}}{\Longrightarrow}} wA\delta$$

と長さ k の先読み u に対して，いつも以下のような生成規則 $B \rightarrow \alpha$ が高々 1 個存在し，かつ $A \overset{*}{\Longrightarrow} B\gamma$ で次のいずれかが成立する．

1. (i) $\alpha = C\beta$ なら， u は $\mathrm{FIRST}_k(\beta\gamma\delta)$ の要素である．
   (ii) さらに C = A なら， u は $\mathrm{FIRST}_k(\delta)$ の要素でない．
2. α が非終端記号で始まらないなら，u は $\mathrm{FIRST}_k(\alpha\gamma\delta)$ の要素である． □

方法 5.14 (LC(k) 解析表構成)

入力: LC(k) 文法 G.

出力: 文法 G に対する LC(k) 構文解析表 T.

手順:

解析スタックの先頭記号と先読みに対し，次のように解析動作を定義する．

1. i 番目の生成規則 $B \rightarrow \alpha$ について
   (i) $\alpha = C\,\beta$ の場合，$T([A, C \rightarrow \tau], x) = (\beta[A, B \rightarrow C\,\beta], i)$ とする．ただし，x は $\mathrm{FIRST}_k(\beta\gamma\delta)$ の要素で，$S \overset{*}{\underset{\mathrm{lc}}{\Longrightarrow}} wA\delta$ で，$A \overset{*}{\Longrightarrow} B\,\gamma$ とする．
   (ii) α が非終端記号で始まらない場合，$T(A, x) = (\alpha[A, B \rightarrow \alpha], i)$ とする．ただし，x は $\mathrm{FIRST}_k(\alpha\gamma\delta)$ の要素で，$S \overset{*}{\underset{\mathrm{lc}}{\Longrightarrow}} wA\delta$ で，$A \overset{*}{\Longrightarrow} B\,\gamma$ とする．
2. $T([A, A \rightarrow \tau], x) = (\varepsilon, \varepsilon)$ とする．ただし x は $\mathrm{FIRST}_k(\delta)$ の要素で，$S \overset{*}{\underset{\mathrm{lc}}{\Longrightarrow}} wA\delta$ で，$A \rightarrow \tau$ は 1 つの A 規則．
3. それぞれの終端記号 a について，T(a, a) = “pop” とする．
4. $T(\$, \varepsilon)$ = “accept”
5. その他の記入項目は “error” とする． □

表 5.29 解析過程

入力	スタック	解析列
.111	S$	ε
.111	.L[S, S → .L]$	1
111	L[S, S → .L]$	1
111	1[L, B → 1][S, S → .L]$	14
11	[L, B → 1][S, S → .L]$	14
11	[L, L → B][S, S → .L]$	143
11	B[L, L → LB][S, S → .L]$	1432
11	1[B, B → 1][L, L → LB][S, S → .L]$	14324
1	[B, B → 1][L, L → LB][S, S → .L]$	14324
1	[L, L → LB][S, S → .L]$	14324
1	B[L, L → LB][S, S → .L]$	143242
1	1[B, B → 1][L, L → LB][S, S → .L]$	1432424
ε	[B, B → 1][L, L → LB][S, S → .L]$	1432424
ε	[L, L → LB][S, S → .L]$	1432424
ε	[S, S → .L]$	1432424
ε	$	1432424
ε	$	1432424a

例 5.17 (LC(1) 構文解析)
例 5.16 の文法は，LC(1) 文法になっている．左再帰の非終端記号 L が存在するので，LL(1) 文法ではない点に注意されたい．LC(1) 構文解析表を作ると，先に示した表 5.28 のようになる．入力 .111 に対する構文解析の過程は表 5.29 のようになる． □

5.5 手続き型構文解析法

手続き型構文解析法は，先に述べた下降型構文解析を手続きを用いて実現する 1 つの方法である．構文解析の仕事を，たくさんの手続きで分業し，お互いに仕事を依頼し合う点に特徴がある．

方法 5.15 (再帰降下型解析)

入力: 文脈自由文法 G と終端記号列 x.

出力: x の左解析列またはエラーメッセージ.

手順:

1. 文法 G の生成規則 $A \to \alpha$，非終端記号 A，終端記号 a に対し，それぞれ手続き $P(A \to \alpha)$，手続き P(A)，手続き P(a) を次のように構成する.

   まず，生成規則 $X_0 \to X_1\ X_2 \cdots X_n$ に対しては，次のような手続きを定義し，

   ```
   procedure P(X0 → X1 X2 ··· Xn) is
   begin
     P(X1); P(X2); ···; P(Xn)
   end.
   ```

   非終端記号 A に対しては，次のような手続きを定義して，

   ```
   procedure P(A) is
   begin
     現在の入力記号と先読み記号列から区別して
     P(A → α1), P(A → α2), ···, P(A → αm)
     のうちの 1 つの手続きを呼ぶ.
   end.
   ```

   終端記号 a に対しては，次のような手続きを定義する.

   ```
   procedure P(a) is
   begin
     入力記号が a でなければ，エラーで異常終了する.
     入力記号が a であれば，記号 a を消費する.
   end.
   ```

2. 開始記号 S に対応した手続き P(S) を呼び出す．途中，終端記号の直接の照合に失敗すれば異常終了となる．手続き P(S) が完了し，その時点の入力列が空列であれば正常終了，空列でなければ異常終了とする．□

例 5.18 (再帰降下型解析)

算術式の認識を言語 C の 4 つの関数 main(), e(), t(), f() に分担して行わせる例を以下に示す．まず文法を，拡張記法の文脈自由文法を用いて，

$$
\begin{aligned}
&S' \to E\\
&E \to T\ [\ +T \mid -\ T]*\\
&T \to F\ [\ *\ T \mid /\ T]*\\
&F \to i \mid (\ E\ )
\end{aligned}
$$

と変形する．そして，プログラム 5.1 のような関数を作る．

プログラム 5.1

```
int next;
main()
{
  scan();
  e();
  if (next != '$') error();
  exit(0);
}
scan()
{
  int i;
  i=scanf("%1s", &next);
  if (i == -1) next ='$';
}
error()
{
  printf("error\n");
  exit(1);
}
e()
{
  t();
  while (next == '+' || next == '-')
  {
```

```
    scan(); t();
  }
}
t()
{
  f();
  while (next == '*' || next == '/')
  {
    scan(); f();
  }
}
f()
{
  if (next >= 'a' && next <= 'z')
    {scan();}
  else
  if (next != '(') error();
  else
  { scan();
    e();
    if (next != ')') error();
    else scan();
  }
}
```

さらに，パラメータを各手続きに 1 つずつ持たせることにより，プログラム 5.2 では，算術式から 4 つ組への翻訳も行うことができる．

プログラム 5.2

```
int next;
int tmp=0;
main()
{
  int y;
  scan();
```

```
  e(&y);
  if (next != '$') error();
  exit(0);
}
scan()
{
  int i;
  i=scanf("%1s",&next);
  if (i == -1) next ='$';
}
newtmp()
{ tmp++;}
error()
{
  printf("error \n");
  exit(1);
}
e(x)
int *x;
{
  int y, z, op;
  t(&y);op=next;
  while (op == '+' || op == '-')
  {
    scan(); t(&z);
    newtmp();
    printf("%c %3d %3d %3d\n", op, y, z, tmp);
    y=tmp;op=next;
  }
  *x=y;
}
t(x)
int *x;
{
  int y, z, op;
  f(&y); op=next;
  while (op == '*' || op == '/')
```

```
  {
    scan(); f(&z);
    newtmp();
    printf("%c %3d %3d %3d\n", op, y, z, tmp);
    y=tmp;op=next;
  }
  *x=y;
}
f(x)
int *x;
{
  int y;
  if (next >= 'a' && next <= 'z')
    {*x=next; scan();}
  else
  if (next != '(') error();
  else
  { scan();
    e(&y); *x=y;
    if (next != ')') error();
    else scan();
  }
}
```

5.6　索表計算型構文解析

表を作りながら構文解析を進める，索表計算型構文解析の代表的方法である CYK 法と Earley 法について述べる．これらはいずれも，一般の文脈自由文法に対し，入力列の長さを n として，n^3 に比例した時間で，認識が終了する．

まず，**CYK**(Cocke–Younger–Kasami) 法を説明する．Chomsky の標準形の文法 G と入力列 $X_1X_2\cdots X_n$ が与えられたとき，入力列 $X_1X_2\cdots X_n$ が L(G) の要素であるかどうかを判定する場合で説明する．Chomsky の標準

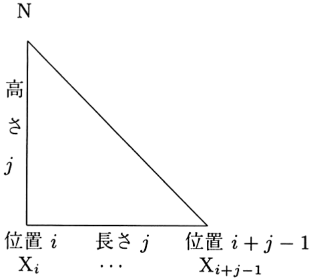


図 5.5 導出木

形では，開始記号のみが持ちうる空規則を除き，右辺が 2 つの非終端記号または 1 つの終端記号となっている．

定義 5.18 (集合 C(j, i))

次のような非終端記号の集合 C(j, i) を考える．ただし，i, j は 1 以上 n 以下で，$i+j-1$ は 1 以上 n 以下とする．

$$\mathrm{N} \overset{*}{\Longrightarrow} \mathrm{X}_i\mathrm{X}_{i+1}\cdots\mathrm{X}_{i+j-1} \quad \overset{\text{定義}}{\Longleftrightarrow} \quad \mathrm{N} \text{ が } \mathrm{C}(j, i) \text{ の要素.}$$ □

C(n, 1) に G の開始記号 S が存在するときおよびそのときに限り，入力列が L(G) の要素であることになる．C(j, i) は 図 5.5 のような，L 字型の 2 辺が，長さ j の高さ，長さ j の底辺となる直角二等辺三角形の導出木で，底辺の左端が入力列の位置 i にあるときの，根に来ることができる非終端記号の集合と考えられる．

集合 C(j, i) は帰納的に計算することができる．

1. $j = 1$ なら

$$\mathrm{N} \overset{*}{\Longrightarrow} \mathrm{X}_i$$

となる N の全体だから，

$$\mathrm{C}(1, i) = \{\ \mathrm{A} \mid \mathrm{A} \rightarrow \mathrm{X}_i\}$$

となる．

2. j が 2 以上 n 以下のとき，k を 1 以上で，規則 A → Y Z が存在して，

$$\mathrm{Y} \overset{*}{\Longrightarrow} \mathrm{X}_i\mathrm{X}_{i+1}\cdots\mathrm{X}_{i+k-1}$$
$$\mathrm{Z} \overset{*}{\Longrightarrow} \mathrm{X}_{i+k}\mathrm{X}_{i+k+1}\cdots\mathrm{X}_{i+j-1}$$

であるなら，

$$A \Rightarrow Y\ Z \Rightarrow X_i X_{i+1} \cdots X_{i+j-1}$$

であるから，

$$C(j,\, i) = \left\{ A \,\middle|\, \begin{array}{l} A \to Y\ Z,\ Y \text{ は } C(k,\, i) \text{ の要素で，} Z \text{ は } C(j-k,\, i+k) \\ \text{の要素} \end{array} \right\}$$

となる．

これを索表計算で行うと，値の依存性を考えて，次のようになる．

方法 5.16 (CYK 法)

入力: Chomsky の標準形の文法 G と入力列 $X = X_1 X_2 \cdots X_n (n \geq 1)$.

出力: X が L(G) の要素なら “yes”，X が L(G) の要素でないなら “no”.

手順:

1. $j=1$ として $i=1$ から n までの値 C(1, 1)，C(1, 2)，$\cdots$，C(1, n) を求める．
2. $j=2$ から n まで，ステップ 3. から 6. を繰り返す．
3. $i=1$ から $n-j+1$ まで，ステップ 4 から 6 を繰り返す．
4. $C(j,\, i) \leftarrow \phi$
5. $k=1$ から $j-1$ まで，ステップ 6. を繰り返す．
6.

$$C(j,\, i) \leftarrow C(j,\, i) \cup \left\{ A \,\middle|\, \begin{array}{l} A \to Y\ Z,\ Y \text{ は } C(k,\, i) \text{ の要素で，} Z \\ \text{は } C(j-k,\, i+k) \text{ の要素} \end{array} \right\}.$$

7. C(n, 1) に G の開始記号 S があれば “yes” を出力する．なければ “no” を出力する．□

例 5.19 (CYK 法)

次のような文法の下で，入力列 aaabb を考える．

j=5	S				
j=4	S	S			
j=3	L	S	S		
j=2	L	L	S	M	
j=1	L, A	L, A	L, A	M, B	M, B
	i=1 a	i=2 a	i=3 a	i=4 b	i=5 b

図 5.6　CYK 法

$$
\begin{array}{ll}
\mathrm{S} \to \mathrm{L\ M} & \mathrm{M} \to \mathrm{M\ B} \\
\mathrm{L} \to \mathrm{L\ A} & \mathrm{M} \to \mathrm{b} \\
\mathrm{L} \to \mathrm{a} & \mathrm{B} \to \mathrm{b} \\
\mathrm{A} \to \mathrm{a} &
\end{array}
$$

索表計算の結果は図 5.6 のようになる．　□

次に，**Earley 法**について述べる．Earley 法では，生成規則 $\mathrm{A} \to \alpha\,\beta$ の右辺の端か文法記号の間にドットを持つ項目 $[\mathrm{A} \to \alpha \cdot \beta]$ を用いる．この項目を **Earley 項目**と呼ぶ．

定義 5.19 (Earley 項目の集合)

文脈自由文法 G，入力列 $\mathrm{w} = \mathrm{a}_1\mathrm{a}_2\cdots\mathrm{a}_m$ に対する Earley 項目の集合 E[G, w](k, i) に，項目 $[\mathrm{A} \to \alpha \cdot \beta]$ が要素として存在するための必要十分条件は，以下の通りである．ただし $0 \leq k \leq i \leq m$ である．E[G, w](k, i) を混乱が起こらない限り E(k, i) と書く．

$\mathrm{A} \to \alpha \cdot \beta$ が E(k, i) の要素であるのは，文形式 $\gamma\mathrm{A}\delta$ が存在して，

$$
\begin{aligned}
&\mathrm{S}' \overset{*}{\Longrightarrow} \gamma\mathrm{A}\delta \Rightarrow \gamma\alpha\beta\delta, \\
&\gamma \overset{*}{\Longrightarrow} \mathrm{a}_1\cdots\mathrm{a}_k, \\
&\alpha \overset{*}{\Longrightarrow} \mathrm{a}_{k+1}\cdots\mathrm{a}_i,
\end{aligned}
$$

となるときおよびそのときに限る．(ただし，$p > q$のとき，$\mathrm{a}_p\cdots\mathrm{a}_q$ は ε を

表すことにする.)　□

すなわち，ある文形式 $\gamma\alpha\beta\delta$ が存在して，γ から $a_1a_2\cdots a_k$ が導出でき，α から $a_{k+1}\cdots a_i$ が導出できる．したがって，$[S' \to S\ \cdot]$ が集合 $E(0, m)$ に属すならば，右辺 S から入力列 $a_1\cdots a_m$ が導出できることになり，入力列が L(G) に属すことになる．

Earley 項目の集合の数え上げは，次の 4 つの規則により帰納的に行うことができる．

1. 開始規則
   0 番目の規則 $S' \to S$ について，項目 $[S' \to \cdot\ S]$ は集合 $E(0, 0)$ の要素とする．
2. ふくらまし規則
   集合 $E(i, j)$ に $[A \to \alpha \cdot B\ \beta]$ があれば，$[B \to \cdot\ \gamma]$ を，すでに存在していない場合，集合 $E(j, j)$ に加える．
3. 非終端記号の引っ越し規則
   集合 $E(k, i)$ に $[A \to \alpha{\cdot}B\beta]$ と集合 $E(i, j)$ に $[B \to \gamma\cdot]$ が共にあれば，$[A \to \alpha B{\cdot}\beta]$ を，すでに存在してしない場合，集合 $E(k, j)$ に加える．
4. 終端記号の引っ越し規則
   集合 $E(i, j-1)$ に項目 $[A \to \alpha{\cdot}a\beta]$ があり，$X_j = a$ であれば，項目 $[A \to \alpha a{\cdot}\beta]$ を，すでに存在していない場合，集合 $E(i, j)$ に加える．

方法 5.17 (Earley 項目の集合の数え上げ)

入力: 文脈自由文法 G と入力列 $w = X_1X_2\cdots X_m$.

出力: 項目の集合 $E(k, i)$ の集合．ただし $0 \le k \le i \le m$ である．

手順:

1. 各集合 $E(k, i)$ を空集合とする．0 番目の規則 $S' \to S$ について，$[S' \to \cdot\ S]$ を集合 $E(0, 0)$ に加える．
2. 次の操作を，新しい項目が加えられなくなるまで繰り返す．
   (i) 集合 $E(0, 0)$ に $[A \to \alpha \cdot B\ \beta]$ と $[B \to \gamma\ \cdot]$ が共にあれば，$[A \to \alpha\ B \cdot \beta]$ を集合 $E(0, 0)$ に加える．
   (ii) 集合 $E(0, 0)$ に $[A \to \alpha \cdot B\ \beta]$ があれば，$[B \to \cdot\ \gamma]$ を集合 $E(0, 0)$

に加える.

3. $t = 1$ から $t = m$ まで以下を順に繰り返す.

(i) $E(0, t-1)$, $E(1, t-1)$, $\cdots$, $E(t-1, t-1)$ に対し, 次の操作を新しい項目が加えられなくなるまで繰り返す.

集合 $E(i, j-1)$ に $[A \to \alpha \cdot a\,\beta]$ があり, $X_j = a$ であれば, $[A \to \alpha\, a \cdot \beta]$ を集合 $E(i, j)$ に加える.

(ii) $E(0, t)$, $E(1, t)$, $\cdots$, $E(t, t)$ に対し, 次の操作を新しい項目が加えられなくなるまで繰り返す.

a. 集合 $E(i, j)$ に $[B \to \gamma\, \cdot]$, 集合 $E(k, i)$ に $[A \to \alpha \cdot B\,\beta]$ があれば, $[A \to \alpha\, B \cdot \beta]$ を集合 $E(k, j)$ に加える.

b. 集合 $E(i, j)$ に $[A \to \alpha \cdot B\,\beta]$ があれば, $[B \to \cdot\, \gamma]$ を集合 $E(j, j)$ に加える. □

例 5.20 (Earley 項目の集合)

次の文法 G と入力列 w = aabb の例で, Earley 項目の集合 E(G, w) を数え上げてみよう.

0) $S' \to S$	5) $A \to b\,A\,A$
1) $S \to a\,B$	6) $B \to b$
2) $S \to b\,A$	7) $B \to b\,S$
3) $A \to a$	8) $B \to a\,B\,B$
4) $A \to a\,S$	

数え上げが終了すると, 図 5.7 のようになる. □

構文解析木を作る場合には, 項目の間に上下関係と右左関係のポインタを接続すればよい. 上下関係のポインタは, 親の節点と子の節点の関係を示し, 右左関係のポインタは, 同一節点であることと, それらの節点の子の節点の年長関係を示す. 右の子が弟で, 左の子が兄である.

以下のように, 4 つの規則のうち, 開始規則とふくらまし規則を除く, 2 つの規則にポインタ接続の動作を追加する.

3. 非終端記号の引っ越し規則

集合 $E(k, i)$ に $[A \to \alpha{\cdot}B\beta]$ と集合 $E(i, j)$ に $[B \to \gamma{\cdot}]$ が共にあれば,

	i=0	i=1	i=2	i=3	i=4
k=0	[S′ → · S] [S → · a B] [S → · b A]	[S → a · B]			[S → a B ·] [S′ → S ·]
k=1		[B → · b] [B → · b S] [B → · a B B]	[B → a · B B]	[B → a B · B]	[B → a B B ·]
k=2			[B → · b] [B → · b S] [B → · a B B]	[B → b ·] [B → b · S]	
k=3				[B → · b] [B → · b S] [B → · a B B] [S → · a B] [S → · b A]	[B → b ·] [B → b · S] [S → b · A]
k=4					[A → · a] [A → · a S] [A → · b A A] [S → · a B] [S → · b A]

図 5.7　Earley 項目の集合

$[\mathrm{A} \to \alpha\mathrm{B}{\cdot}\beta]$ を集合 $\mathrm{E}(k, j)$ に加える．

このとき，上下関係のポインタを $\mathrm{E}(k, j)$ の $[\mathrm{A} \to \alpha\ \mathrm{B} \cdot \beta]$ から $\mathrm{E}(i, j)$ の $[\mathrm{B} \to \gamma\ \cdot]$ へ接続し，右左関係のポインタを， $\mathrm{E}(k, j)$ の $[\mathrm{A} \to \alpha\ \mathrm{B} \cdot \beta]$ から $\mathrm{E}(k, i)$ の $[\mathrm{A} \to \alpha \cdot \mathrm{B}\ \beta]$ へ接続する．

4. 終端記号の引っ越し規則

集合 $\mathrm{E}(i, j-1)$ に項目 $[\mathrm{A} \to \alpha \cdot \mathrm{a}\ \beta]$ があり，$\mathrm{X}_j = \mathrm{a}$ であれば，項目 $[\mathrm{A} \to \alpha\ \mathrm{a}\ {\cdot}\beta]$ を集合 $\mathrm{E}(i, j)$ に加える．

このとき，右左関係のポインタを，$\mathrm{E}(i, j)$ の $[\mathrm{A} \to \alpha\mathrm{a}{\cdot}\beta]$ から $\mathrm{E}(i, j-1)$ の $[\mathrm{A} \to \alpha{\cdot}\mathrm{a}\beta\ ]$ へ接続する．

項目 $[\mathrm{S}' \to \mathrm{S}\ \cdot]$ が集合 $\mathrm{E}(0, m)$ に属すならば，項目 $[\mathrm{S}' \to \mathrm{S}\ \cdot]$ は構文解析木全体の根である．一般には多くの構文解析木をこれらのポインタから作ることができる．これらの構文解析木から 1 つの右解析列を出力する場合の方法を以下に示す．記述を簡潔にするために集合 $\mathrm{E}(i, j)$ に属す項目 $[\mathrm{A} \to \alpha \cdot \beta]$ を，明示的に 3 つ組 $[\mathrm{A} \to \alpha \cdot \beta, i, j]$ と書く．

方法 5.18 (Earley 法による右解析列の出力)

入力: 文脈自由文法 G，入力列 $\mathrm{w} = \mathrm{X}_1\mathrm{X}_2\cdots\mathrm{X}_m$ と 3 つ組の集合 E(G, w).

出力: 1 つの右解析列 π またはエラーメッセージ.

手順:

1. 3 つ組 $[\mathrm{S}' \to \mathrm{S}\ \cdot, 0, m]$ が集合 E(G, w) に属さなければ，エラーメッセージを出力する．3 つ組 $[\mathrm{S}' \to \mathrm{S}\ \cdot, 0, m]$ が集合 E(G, w) に属すならば， π を空列にして，次の手続き $\mathrm{P}([\mathrm{S}' \to \mathrm{S}\ \cdot, 0, m])$ を呼ぶ.
2. 手続き $\mathrm{P}([\mathrm{A} \to \alpha\ \cdot, i, j])$ の定義
   (i) 規則 $\mathrm{A} \to \alpha$ の番号に現在の π をつなげた結果を，π とする．
   (ii) $\alpha = \mathrm{Y}_1\ \mathrm{Y}_2 \cdots \mathrm{Y}_n$ とする．$k = n$，$l = j$ とする．
   (iii) ステップ (iv) を $k = 0$ となるまで繰り返す.
   (iv) a. Y_k が終端記号なら，k も l も 1 減らす.
   b. Y_k が非終端記号なら，次のような r を 1 つ見つける．
   $[\mathrm{A} \to \mathrm{Y}_1 \cdots \mathrm{Y}_{k-1} \cdot \mathrm{Y}_k \cdots \mathrm{Y}_n, i, r]$ も $[\mathrm{Y}_k \to \gamma\ \cdot, r, l]$
   も共に E(G, w) に属す.
   手続き $\mathrm{P}([\mathrm{Y}_k \to \gamma\ \cdot, r, l])$ を呼ぶ.

k を 1 減らし，$l = r$ とする．□

例 5.21 (右解析列作成)

例 5.20 の文法と入力列に対する Earley 項目の集合 E(G, w) を用いて，入力列 aabb に対する 1 つの右解析列を求めてみよう．方法 5.18 による実行の様子は次のようになる．

1. 最初，$\pi = \varepsilon$ である．
2. $\pi = 0$ となる．3 つ組 [S′ → S ·, 0, 4] は，3 つ組 [S′ → · S, 0, 0] と [S → a B ·, 0, 4] に分解される．
3. $\pi = 10$ となる．3 つ組 [S → a B ·, 0, 4] は，3 つ組 [S → a · B, 0, 1] と [B → a B B ·, 1, 4] に分解される．
4. $\pi = 810$ となる．3 つ組 [B → a B B ·, 1, 4] は，まず 3 つ組 [B → a B · B, 1, 3] と [B → b ·, 3, 4] に分解される．
5. $\pi = 6810$ となる．3 つ組 [B → b ·, 3, 4] は，これ以上分解できない．
6. 先の 3 つ組 [B → a B B ·, 1, 4] は，次に [B → a · B B, 1, 2] と [B → b ·, 2, 3] に分解される．
7. $\pi = 66810$ となる．3 つ組 [B → b ·, 2, 3] は，これ以上分解できない．
8. 終了．□

第6章　構文解析の理論

文脈自由言語を認識する装置であるプッシュダウンオートマトンのモデル，非決定性プッシュダウンオートマトンの受理言語のクラスと文脈自由言語のクラスの同一性，文脈自由言語の特徴付け，構文解析の方法を支える基本的理論について述べる．

6.1　プッシュダウンオートマトン

文脈自由言語を受理する機械のモデルである，非決定性プッシュダウンオートマトンと決定性プッシュダウンオートマトンの 2 種類のモデルについて述べる．これらのモデルは言語の認識や構文解析の過程を抽象化することによって得られたモデルである．

非決定性プッシュダウンオートマトンとは，ε 遷移を許す非決定性有限オートマトンにプッシュダウンスタックを持たせ，次のような能力強化を行った機械である．

現在の自分の状態，入力記号，プッシュダウンスタックの先頭記号を見て，あるいは現在の自分の状態，プッシュダウンスタックの先頭記号のみを見て，次の状態を決め，スタックの先頭記号をスタック記号列に置き換えることができる．

定義 6.1 (非決定性プッシュダウンオートマトン)
非決定性プッシュダウンオートマトン $\mathrm{M} = (\mathrm{Q}, \mathrm{T}, \mathrm{V}, \delta, i, \mathrm{A}, \mathrm{F})$ とは，次の構成要素からなる7つ組である．

1. 状態の有限集合 Q.
2. 入力記号の有限集合 T.
3. スタック記号の有限集合 V.
4. 状態遷移関数 δ.
5. Q の1つの要素である開始状態 i.
6. V の1つの要素である開始スタック記号 A.
7. Q の部分集合である最終状態の集合 F.

ここで，状態遷移関数 δ は， Q の要素 q， V の要素 v， T の要素 a または ε に対し， Q の要素とスタック記号の長さ0以上の記号列の2つ組の有限集合を対応させる規則である．すなわち，状態遷移関数 δ は次のような一般形を持つ．

$$\delta(q, v, \mathrm{a}) = \{(p_1, \mathrm{x}_1), \cdots, (p_n, \mathrm{x}_n)\},$$
$$\delta(q, v, \varepsilon) = \{(r_1, \mathrm{y}_1), \cdots, (r_m, \mathrm{y}_m)\}.$$ □

定義 6.2 (決定性プッシュダウンオートマトン)
決定性プッシュダウンオートマトンとは，次のような性質を持つ状態遷移関数 δ を持つ非決定性プッシュダウンオートマトンのことである．

1. $\delta(q, v, \mathrm{a})$ は高々1個の2つ組を要素に持つ集合である．
2. $\delta(q, v, \varepsilon)$ は高々1個の2つ組を要素に持つ集合である．
3. $\delta(q, v, \varepsilon)$ が非空集合のとき, $\delta(q, v, \mathrm{a})$ はどの a についても空集合である． □

定義 6.3 (非決定性プッシュダウンオートマトンの受理言語)

1. 状態 q, スタック記号列 α, 入力記号列 w の3つ組 (q, α, w) を，プッシュダウンオートマトンの**計算状況**と呼ぶ．
2. プッシュダウンオートマトンの計算状況の上の遷移関係 $\vdash$ を，次のように定義する．

$(q_1, \alpha_1, \mathrm{w}_1) \vdash (q_2, \alpha_2, \mathrm{w}_2) \overset{\text{定義}}{\Longleftrightarrow}$ $\mathrm{w}_1 = \mathrm{X}\cdot\mathrm{w}_2, \alpha_1 = \mathrm{V}\cdot\alpha, \alpha_2 = \gamma\cdot\alpha$, (q_2, γ) は $\delta(q_1, \mathrm{V}, \mathrm{X})$ の要素, X は ε または終端記号 1 個.

3. 非決定性プッシュダウンオートマトン $\mathrm{M} = (\mathrm{Q}, \mathrm{T}, \mathrm{V}, \delta, i, \mathrm{A}, \mathrm{F})$ の受理する言語 L(M) を，次のように定義する．

$$\mathrm{L(M)} = \{ \mathrm{w} \mid (i, \mathrm{A}, \mathrm{w}) \vdash * (q, \alpha, \varepsilon), q \text{ は F の要素} \}. \qquad \square$$

プッシュダウンオートマトンのこの受理言語の定義の仕方を，**終了状態受理方式**と呼ぶ．受理言語の定義の仕方にはもう 1 つの流儀がある．スタックの内容が空になると，次の動作が存在せず，機械が停止してしまうことを利用する空スタック受理方式である．

定義 6.4 (空スタック受理方式の受理言語)
非決定性プッシュダウンオートマトンの空スタック受理方式の受理言語を，次のよう定義する．

非決定性プッシュダウンオートマトン $\mathrm{M} = (\mathrm{Q}, \mathrm{T}, \mathrm{V}, \delta, i, \mathrm{A}, \phi)$ の空スタック受理方式の受理言語 $\mathrm{L}\phi(\mathrm{M})$ を，次のように定義する．

$$\mathrm{L}\phi(\mathrm{M}) = \{ \mathrm{w} \mid (i, \mathrm{A}, \mathrm{w}) \vdash * (q, \varepsilon, \varepsilon)\}. \qquad \square$$

終了状態受理方式でも空スタック受理方式でも，言語を受理する能力に違いがないことを示す．

定理 6.1 (空スタック受理方式の受理言語)
1. 言語 L が終了状態受理方式の非決定性プッシュダウンオートマトンで受理できるならば，空スタック受理方式の非決定性プッシュダウンオートマトンでも受理できる．
2. 言語 L が空スタック受理方式の非決定性プッシュダウンオートマトンで受理できるならば，終了状態受理方式の非決定性プッシュダウンオートマトンでも受理できる． □

証明
1. 非決定性プッシュダウンオートマトン $\mathrm{M} = (\mathrm{Q}, \mathrm{T}, \mathrm{V}, \delta, i, \mathrm{A}, \mathrm{F})$ に新し

い状態 q_0 を加え，各終了状態 q に対し，$\delta(q, \mathrm{v}, \varepsilon) = \{(q_0, \mathrm{v})\}$ とする．そして $\delta(q_0, \mathrm{v}, \varepsilon) = \{(q_0, \varepsilon)\}$ とする．つまり本来の終了状態に，入力列すべてを消費して到達したのならば，新しい状態 q_0 へ行き，ことごとくスタック記号を取り去って行くのである．

2. 空スタック受理方式の非決定性オートマトン $\mathrm{M} = (\mathrm{Q}, \mathrm{T}, \mathrm{V}, \delta, i, \mathrm{A}, \phi)$ に新しい状態 q_1 と q_2 を加え，新しい開始状態を q_1，新しい開始スタック記号を A_1 として，遷移 $\delta(q_1, \mathrm{A}_1, \varepsilon) = \{(i, \mathrm{A}\ \mathrm{A}_1)\}$ とする．
   スタックが空になったとき，すなわちスタックの内容が A_1 のみとなったとき，q_0 以外の任意の q について，遷移 $\delta(q, \mathrm{A}_1, \varepsilon) = \{(q_2, \mathrm{A}_1)\}$ とする．そして状態 q_2 を終了状態とする．□

6.2　文脈自由文法と非決定性文脈自由言語

非決定性プッシュダウンオートマトンの受理言語のクラスと文脈自由文法の生成する言語のクラスが，一致することを示す．

定理 6.2 (文脈自由言語との関係)

1. 言語 L が文脈自由文法 G で生成できるならば，空スタック受理方式の非決定性プッシュダウンオートマトン M で受理できる．
2. 言語 L が空スタック受理方式の非決定性プッシュダウンオートマトン M で受理できるならば，文脈自由文法 G で生成できる．□

証明

1. 文脈自由文法 $\mathrm{G} = (\mathrm{N}, \mathrm{T}, \mathrm{P}, \mathrm{S})$ の各生成規則から，次のような遷移を持った空スタック受理方式の非決定性プッシュダウンオートマトン $\mathrm{M} = (\{q\}, \mathrm{T}, \mathrm{N} \cup \mathrm{T}, \delta, q, \mathrm{S}, \phi)$ を作る．ただし，$\mathrm{F} = \{[q]\}$ とする．
   $\mathrm{A} \rightarrow \mathrm{X}_1\ \mathrm{X}_2 \cdots \mathrm{X}_n$ から，$(q, \mathrm{X}_1\ \mathrm{X}_2 \cdots \mathrm{X}_n)$ を $\delta(q, \mathrm{A}, \varepsilon)$ の要素とする．また終端記号 a について，(q, ε) を $\delta(q, \mathrm{a}, \mathrm{a})$ の要素とする．
2. 空スタック受理方式の非決定性プッシュダウンオートマトン $\mathrm{M} = (\mathrm{Q}, \mathrm{T}, \mathrm{V}, \delta, i, \mathrm{A}, \phi)$ から次のような文脈自由文法の生成規則を作る．

(この文脈自由文法から導出される文形式は消費済みの入力記号列とその時点のスタック記号列を表現している．例えば，文形式 a_1a_2 $[s_0, X_1, s_1]$ $[s_1, X_2, s_2]$ は消費済みの入力記号列が a_1a_2 で，その時点のスタック記号が X_1X_2 であることを示す．また，記号 $[s_0, X_1, s_1]$ はスタックに記号 X_1 をプッシュしたときの状態が s_0, X_1 をポップするときの状態が s_1 であることを示す.)

まず Q の要素 q について，$S \to [i, A, q]$ を作る．

(i) $(r, X_1 \cdots X_k)$ が $\delta(p, X, a)$ の要素の場合

$$[p, X, s_k] \to a[r, X_1, s_1] \cdots [s_{k-1}, X_k, s_k].$$

(ii) (r, ε) が $\delta(p, X, a)$ の要素の場合

$$[p, X, r] \to a.$$

(iii) $(r, X_1 \cdots X_k)$ が $\delta(p, X, \varepsilon)$ の要素の場合

$$[p, X, s_k] \to [r, X_1, s_1] \cdots [s{-}k-1, X_k, s_k].$$

(iv) (r, ε) が $\delta(p, X, \varepsilon)$ の要素の場合

$$[p, X, r] \to \varepsilon.$$

□

6.3　文脈自由言語の特徴付け

部屋割り論法を用いて，文脈自由言語についてのポンプの補題を示す．

定理 6.3 (文脈自由言語のポンプの補題)

文脈自由言語 L に対して，ある整数 k が存在して，長さ k 以上のどんな L の文 z も，次の性質を満たす．

1. z = uvwxy と書ける．
2. vx の長さは 1 以上である．
3. vwx の長さは 1 以上 k 以下である．
4. すべての 0 以上の整数 i に対し，uv^iwx^iy は L の文である．　□

証明

言語 L を生成する Chomsky の標準形の 1 つの文脈自由文法 G を考える．文法 G の非終端記号の個数を n とする．$k = 2^n$ とおく．長さ 2^n 以上の文

z を考える．z の長さは $2^{n-1}+1$ 以上なので，z の導出木の高さは $n+1$ 以上である．(そうでなければ，z の長さは 2^{n-1} 以下になってしまう.) したがって，少なくとも長さ $n+1$ 以上のパスで終端記号 X_0 から開始記号 X_{m+1} まで，途中 m 個の非終端記号を通過する次のような列が，1 つある $(n \leq m)$.

$$\mathrm{X}_0, \mathrm{X}_1, \cdots, \mathrm{X}_m, \mathrm{X}_{m+1}.$$

X_1 から X_{m+1} までの $m+1$ 個は，$n+1$ 個以上なので，少なくとも 1 つの非終端記号の対 X_i と X_j ($1 \leq i < j \leq n+1 \leq m+1$) は，同一の非終端記号となる．そこで z を次のような u, v, w, x, y に分解する．

$$\mathrm{S} \stackrel{*}{\Longrightarrow} \mathrm{u}\ \mathrm{X}_j\ \mathrm{y},$$
$$\mathrm{u}\ \mathrm{X}_j\ \mathrm{y} \stackrel{+}{\Longrightarrow} \mathrm{u\ v}\ \mathrm{X}_i\ \mathrm{x\ y} \stackrel{*}{\Longrightarrow} \mathrm{u\ v\ w\ x\ y} = \mathrm{z}.$$

Chomsky の標準形でしかも $i < j$ なので，v と x が両方ともに ε となることはない．

非終端記号 X_j から文 vwx に至る導出木の部分木の高さは，高々 $n+1$ なので，vwx の長さは 2^n 以下である．X_j と X_i が同一なので，uwy, uvwxy, uvvwxxy, $\cdots$ はすべて L の文となる．□

定理 6.4 (非文脈自由言語)
次の言語を生成する文脈自由文法は，存在しない．

$$\mathrm{L} = \{\mathrm{a}^n\mathrm{b}^n\mathrm{c}^n \mid n \geq 1\}.$$

□

証明
もし L が文脈自由言語であるとすると，定理 6.3 の性質を満たす整数 k が存在する．そこで，長さ k 以上である文 $\mathrm{a}^k\mathrm{b}^k\mathrm{c}^k$ を考える．$\mathrm{a}^k\mathrm{b}^k\mathrm{c}^k$ は uvwxy と書けることになる．vwx の長さは 1 以上 k 以下である．vwx の 長さは k 以下であるので，a と b と c の 3 種類の記号をすべて持つことはできない． vwx は少なくともある 1 種類の記号 (例えば c としよう) は含まない． uvwxy と uwy には同数の a と b と c を含み， uvwxy は uwy より真に長

いので，vx は 1 個以上の a と b と c を同数含むことになる．しかし vx は vwx に c が存在しないので，c を含まない．これは矛盾である．よって L は文脈自由言語でない． □

同様に文脈自由言語のポンプの補題より，次の定理が成立する．これらの結果から，正しい文の長さがきわめて飛び離れている言語は文脈自由言語にならないこと，あるいは，文脈自由文法では文中の離れた部分の個数の一致をとることができないことがわかる．

定理 6.5 (非文脈自由言語)

次の言語を生成する文脈自由文法は，存在しない．

1. $\mathrm{L}_1 = \{\mathrm{a}^{n^2} \mid n \geq 1\}$.
2. $\mathrm{L}_2 = \{\mathrm{a}^{2^n} \mid n \geq 1\}$.
3. $\mathrm{L}_3 = \{\mathrm{a}^m\mathrm{b}^n\mathrm{a}^m\mathrm{b}^n \mid m \geq 1,\, n \geq 1\}$.
4. $\mathrm{L}_4 = \{\mathrm{ww} \mid$ w は a と b からなる長さ 1 以上の列 $\}$. □

定理 6.6 (正規言語との共通集合)

記号集合 T の上の文脈自由言語 L と正規言語 R の共通集合は，文脈自由言語である． □

証明

文脈自由言語 L を受理する非決定性プッシュダウンオートマトンを $\mathrm{M}_1 = (\mathrm{Q}_1, \mathrm{T}, \mathrm{V}, \delta_1, i_1, \mathrm{A}, \mathrm{F}_1)$ とする．正規言語 R を受理する非決定性有限オートマトンを $\mathrm{M}_2 = (\mathrm{Q}_2, \mathrm{T}, \delta_2, i_2, \mathrm{F}_2)$ とする．新しい非決定性プッシュダウンオートマトン $\mathrm{M} = (\mathrm{Q}, \mathrm{T}, \mathrm{V}, \delta, i, \mathrm{A}, \mathrm{F})$ を次のように作る．Q は M_1 の状態 r と M_2 の状態 s の 2 つ組 (r, s) すべての集合とする．M の開始状態 i を (i_1, i_2) とする．

$$\delta((r,\, s),\, \mathrm{v},\, \varepsilon) = \{((p,\, s),\mathrm{x}) \mid \delta_1(r,\, \mathrm{v},\, \varepsilon) \text{ は } (p,\, \mathrm{x}) \text{ を持つ.}\}$$

$$\delta((r,\, s),\, \mathrm{v},\, \mathrm{a}) = \left\{ ((p,\, t),\, \mathrm{x}) \,\middle|\, \begin{array}{l} \delta_1(r,\, \mathrm{v},\, \mathrm{a}) \text{ は } (p,\, \mathrm{x}) \text{ を持つ．} \delta_2(s,\, \mathrm{a}) \\ \text{は } t \text{ を持つ.} \end{array} \right\}$$

F を F_1 の状態 r と F_2 の状態 s の 2 つ組 (r, s) すべての集合とする．□

定義 6.5 (準同型写像による像言語)

T_1，T_2 を 2 つの終端記号集合とする．T_1* から T_2* への写像 h が次の性質を満たすとき，**準同型写像**という．

$$h(\varepsilon) = \varepsilon,$$
$$h(\mathrm{a}_i) = \mathrm{w}_i \ (\mathrm{a}_i \text{ は } \mathrm{T}_1 \text{ の記号，} \mathrm{w}_i \text{ は } \mathrm{T}_2 \text{ の記号列}),$$
$$h(\mathrm{x}\cdot\mathrm{a}) = h(\mathrm{x})\cdot h(\mathrm{a}).$$

T_1 の上の文脈自由言語 L の**像言語** $h(\mathrm{L})$ とは，次の言語のことである．

$$h(\mathrm{L}) = \{h(\mathrm{x}) \mid \mathrm{x} \text{ は L の要素}\}.$$ □

定理 6.7 (像言語)

終端記号 T_1 から終端記号 T_2 への写像 h を，準同型写像とする．T_1 の上の文脈自由言語 L の h による像言語 $h(\mathrm{L})$ は，T_2 の上の文脈自由言語である．□

証明

L を生成する文法の生成規則における終端記号の a_i の出現箇所を，すべて新しい非終端記号 $[\mathrm{a}_i]$ に取り替える．そして新しい生成規則

$$[\mathrm{a}_i] \rightarrow \mathrm{w}_i$$

を付け加えれば，像言語 h(L) を生成する文脈自由文法となる．□

決定性文脈自由言語については，次の定理が知られている．

定理 6.8 (決定性文脈自由言語の補集合)

決定性文脈自由言語 L の補集合は，決定性文脈自由言語となる．□

非決定性文脈自由言語については，次の性質が成立する．

定理 6.9 (文脈自由言語の補集合)

その補集合が文脈自由言語でない，文脈自由言語が存在する．□

証明

$L = \{a^i b^j c^k \mid i \neq j$ または $j \neq k, i \geq 0, j \geq 0, k \geq 0\} \cup (L(a{*}b{*}c{*})^c)$ は容易に文法が作れる文脈自由言語である．一方， L の補集合

$$\{a^i b^j c^k \mid i = j = k, i \geq 0, j \geq 0, k \geq 0\}$$

は，文脈自由言語でない．□

したがって，以下の結論が得られる．

定理 6.10 (非決定性文脈自由言語)

決定性文脈自由言語でない，非決定性文脈自由言語が存在する．□

証明

非決定性プッシュダウンオートマトンが受理する言語のクラスは，補集合をとる操作に対し，閉じていない．一方，決定性プッシュダウンオートマトンが受理する言語のクラスは，補集合をとる操作に対し，閉じている．したがって，特に，次の言語は決定性言語でない．

$$L = \{a^i b^j c^k \mid i \neq j \text{ または } j \neq k, i \geq 0, j \geq 0, k \geq 0\} \cup (\ L(a{*}b{*}c{*})^c\ )$$

もし，決定性言語ならば，その補集合

$$\{a^i b^j c^k \mid i = j = k, i \geq 0, j \geq 0, k \geq 0\}$$

は，文脈自由言語となってしまう．□

決定性文脈自由言語のクラスは，LR(1) 文法の生成する言語のクラスと一致する．また決定性文脈自由言語 L に対し， L に含まれない記号 \$ を各文末に付加して得られる言語 L\$ のクラスは，LR(0) 文法の生成する言語のクラスと一致することが知られている．

6.4　非文脈自由言語

プログラミング言語や自然言語の字句記号の並べ方の規則を，文脈自由文法で完全に表現できればとても便利であるが，残念なことに，文脈自由文法では，例えば，以下のような性質を表現できないことが知られている．

1. 宣言済みの変数のみが，プログラム中で，使用されなければならない．
2. 同一有効範囲内で，同じ変数名が二重に宣言されてはならない．
3. 手続きの宣言時の引数の数は，使用時の引数の数と，一致しなければならない．

この理由を説明してみよう．まず，6.3 節の議論の結果として得られる次の定理を利用する．文脈自由言語と正規言語の共通集合の，準同型写像による像言語は，文脈自由言語である．

いま，正しい PASCAL プログラムの全体を言語 L_0 とし，L_0 は文脈自由言語と仮定する．そして例えば，次のような形の正しい PASCAL プログラムにおいて，w を c と f の長さ 1 以上の文字列としたときの，プログラムの全体を言語 L_1 とする．(w 以外の場所に c も f も出現していないことに注意されたい.)

```
program ex(input, output);
  var w : integer;
begin
  w := 1
end.
```

そして，次のような正規言語 R を考える．

```
program ex(input, output);
  var [c と f の長さ 0 以上の任意の文字列] : integer;
begin
  [c と f の長さ 0 以上の任意の文字列] := 1
end.
```

言語 L_0 とこの正規言語 R との共通集合は L_1 である．L_1 を，次の準同型写像 f で変換すると，

$f(\mathrm{c}) = \mathrm{a}$,
$f(\mathrm{f}) = \mathrm{b}$,
$f(\mathrm{X}) = \varepsilon$ (X は c と f 以外の文字),

L_1 の像言語は文脈自由言語となる．

しかしながら，この集合

{ ww | w は a と b からなる長さ 1 以上の列 }

は，文脈自由言語ではないことが知られている．これは矛盾である．したがって，正しい PASCAL のプログラムの集合 L_0 は，文脈自由言語ではない．

文脈自由言語以外の言語の世界の階層をみておこう．言語の要素であるかどうかを判定する手続きで，“YES” の場合でも “NO” の場合でも必ず停止する手続きが存在する言語を，帰納的言語といい，“YES” の場合だけ停止が保証されるような手続きが存在する言語を帰納的可算言語という．

定義 6.6 (帰納的言語)

言語 L を認識する手続きで，次の性質を満たす手続き P が存在するとき，言語 L を**帰納的言語** (recursive language) という．

1. 与えられた記号列 x が L の要素のとき，停止して “YES” と出力する．
2. 与えられた記号列 x が L の要素でないとき，停止して “NO” と出力する． □

定義 6.7 (帰納的可算言語)

言語 L を認識する手続きで，次の性質を満たす手続き P が存在するとき，言語 L を**帰納的可算言語** (recursively enumerable language) という．

1. 与えられた記号列 x が L の要素のとき，停止して “YES” と出力する．
2. 与えられた記号列 x が L の要素でないとき，停止して “NO” と出力する，あるいは停止しない． □

帰納的可算言語に対しては，言語の要素を順番に列挙し続ける手続きを作ることができる．すなわちある手続きにより，L のどの要素もいつか列挙され，L の要素でないものは，決して列挙されないようにできる．

文脈依存言語のクラスは，帰納的言語のクラスの真の部分クラスとなっている．また帰納的言語でない帰納的可算言語が存在するため，帰納的言語のクラスは，帰納的可算言語のクラスの真の部分クラスとなっている．0 型言語のクラスは，帰納的可算言語のクラスと一致するが，帰納的可算言語でない言語も存在する．

定理 6.11 (非文脈依存言語)
終端記号 T の上で文脈依存言語でない，帰納的言語が存在する．　□

証明
文脈依存文法に標準的順序を与え，すべての文脈依存文法を G_1，G_2，G_3，$\cdots$ と列挙する手続きを用意する．すべての有限入力列を w_1，w_2，w_3，$\cdots$ と標準的順番で列挙する手続きを用意する．次の集合 L_0 を考える．

$$L_0 = \{\ w_i \mid w_i\ \text{は}\ L(G_i)\ \text{の要素でない．}\ \}$$

L_0 は帰納的言語である．なぜなら，与えられた w に対し，$w = w_i$ となる i を計算し，G_i を作り，w_i が $L(G_i)$ の要素であるかどうかを調べればよいからである．この計算はすべて有限時間で停止する．

ところが $L_0 = L(G_j)$ となる文脈依存文法 G_j は存在しない．なぜなら，そのような G_j があるとして，w_j を L_0 の要素と考えると，w_j は $L(G_j) = L_0$ の要素でないことになる．w_j を L_0 の要素でないと考えると，w_j は $L(G_j) = L_0$ の要素であることになり，矛盾が起こる．　□

6.5　LL 構文解析の理論的基礎

LL(k) 構文解析を支える理論的基礎について例を用いて説明する．Wood 法および Tokuda 法は，Lewis-Stearns 法に比べ文脈の区別を，それぞれ非終端記号に関して粗くしたもの，非終端記号については変更せず終端記号に

関して細かくしたものとみなせるので，Lewis-Stearns 法について述べる．

方法 5.1 の素朴式下降型解析で問題になるのは，展開時の右文脈である．ある固定の長さの右文脈で，生成規則の区別がつけば，どの規則で展開すべきかが決定できる．つまり，スタックの先頭が非終端記号 A のとき，A 規則のどれで展開すべきかが，残りの入力列の長さ k の先読みから区別できればよいのである．

$k = 1$ の場合の例で見てみよう．

$$
\begin{array}{ll}
\mathrm{S}' \to \mathrm{S} & \mathrm{A} \to \mathrm{a} \\
\mathrm{S} \to \mathrm{A} & \mathrm{B} \to \mathrm{b} \\
\mathrm{S} \to \mathrm{B} &
\end{array}
$$

この文法の場合，スタックの先頭に S が来たとき，2 つの S 規則のうちのいずれを選ぶかは，長さ 1 の先読みで区別がつく．すなわち，先読みが a ならば，規則 S → A による展開であり，先読みが b ならば，規則 S → B による展開である．このことを，文法記号を見ただけで区別できるように，文法を書き換えてみよう．

$$
\begin{array}{l}
[\mathrm{S}':\mathrm{a, b}] \to [\mathrm{S}:\mathrm{a, b}] \\
[\mathrm{S}:\mathrm{a}] \to [\mathrm{A}:\mathrm{a}] \\
[\mathrm{S}:\mathrm{b}] \to [\mathrm{B}:\mathrm{b}] \\
[\mathrm{A}:\mathrm{a}] \to [\mathrm{a}:\mathrm{a}] \\
[\mathrm{B}:\mathrm{b}] \to [\mathrm{b}:\mathrm{b}]
\end{array}
$$

ここで, 例えば新しい非終端記号 [S′:a, b] は，S′ を展開するのは先読みが a か b かのときであり，新しい終端記号 [a:a] が入力列と一致するのは，先読みが a のときのみであることを示している．

2 つの S 規則はこのように書き換えると，次のように別の非終端記号 [S:a], [S:b] に関する書換え規則となり，区別をつけることができる．

$$
\begin{array}{l}
[\mathrm{S}:\mathrm{a}] \to [\mathrm{A}:\mathrm{a}] \\
[\mathrm{S}:\mathrm{b}] \to [\mathrm{B}:\mathrm{b}]
\end{array}
$$

次に，もう少し複雑な次の例で考えてみる．

$$\begin{array}{ll} S' \to S & S \to B \\ S \to A\ c & A \to a \\ S \to a\ B & B \to b \end{array}$$

この文法の場合，同様の書換えを行うと，次のようになる．

[S′:a, b] → [S:a, b]
[S:a] → [A:a] [c:c]
[S:a] → [a:a] [B:b]
[S:b] → [B:b]
[A:a] → [a:a]
[B:b] → [b:b]

ところが, 元の S 規則 3 つのうち最初の 2 つの新しい非終端記号は,

[S:a] → [A:a] [c:c]
[S:a] → [a:a] [B:b]
[S:b] → [B:b]

[S:a] で一致してしまい，規則の区別がつかない．そこで，長さ 2 の先読み情報を持たせると，元の S 規則は次のように新しい左辺がみな異なる．

[S′:ac, ab, b] → [S:ac, ab, b]
[S:ac] → [A:ac] [c:c]
[S:ab] → [a:ab] [B:b]
[S:b] → [B:b]
[A:a] → [a:a]
[B:b] → [b:b]

つまり，もともとの文法記号に，長さ k の右文脈の集合を持たせて，文法を書き換え，左辺がことごとく異なる非終端記号にしてしまおうとするのが Lewis-Stearns の LL(k) 構文解析である．

最後に，LL(1) 文法が強 LL(1) 文法となる理由を説明しておく．いま，A 規則を，$A \to \alpha$，$A \to \beta$，$\cdots$，$A \to \gamma$ とする．強 LL(1) 文法では，$\mathrm{FOLLOW}_1(A)$ で 2 つの生成規則の展開時の右文脈が分離できるかテスト

している．つまり任意の A 規則 A $\rightarrow \alpha$ と A $\rightarrow \beta$ に関し，以下が共通要素を持たないかどうかをテストしている．

$$\mathrm{FIRST}_1(\alpha) \oplus_k \mathrm{FOLLOW}_1(\mathrm{A}),$$
$$\mathrm{FIRST}_1(\beta) \oplus_k \mathrm{FOLLOW}_1(\mathrm{A}).$$

これに対し，LL(1) 文法では，局所的 FOLLOW 集合 L_i で 2 つの生成規則の展開時の右文脈が共通要素を持たないかどうかをテストしている．

$$\mathrm{FIRST}_1(\alpha) \oplus_k \mathrm{L}_i,$$
$$\mathrm{FIRST}_1(\beta) \oplus_k \mathrm{L}_i.$$

ここで局所的 FOLLOW 集合 L_1，L_2，$\cdots$，L_n 全部の和集合は $\mathrm{FOLLOW}_1(\mathrm{A})$ に一致する．

起こりうるケースは以下の 3 つの場合が考えられる．

1. どの $\mathrm{FIRST}_1(\alpha)$ も ε を持たない場合
   局所的 FOLLOW 集合 L_i も $\mathrm{FOLLOW}_1(\mathrm{A})$ も展開時の右文脈に全く影響を与えないないので，展開時の右文脈は同じように分離でき，題意成立．
2. 1 つの $\mathrm{FIRST}_1(\alpha)$ のみ ε を持ち，その他の $\mathrm{FIRST}_1(\beta)$ はすべて ε を持たない場合
   LL(1) なので，ε を持たない $\mathrm{FIRST}_1(\beta)$ とどの局所的 FOLLOW 集合 L_i とも共通部分はない．したがって，局所的 FOLLOW 集合 L_1，L_2，$\cdots$，L_n 全部の和集合である $\mathrm{FOLLOW}_1(\mathrm{A})$ とも共通部分はない．よって展開時の右文脈が分離でき，題意成立．
3. 少なくとも 2 つの $\mathrm{FIRST}_1(\alpha)$ と $\mathrm{FIRST}_1(\beta)$ が ε を持つ場合
   そもそも LL(1) でなくなってしまうので，題意成立．

以上から LL(1) 文法は強 LL(1) 文法となることが示せた．

6.6 LR構文解析の理論的基礎

LR(k) 構文解析を支える理論的基礎について説明する．Knuth 型以外のLR構文解析表は，Knuth 型LR構文解析表に対し，解析動作の決定性を損なわない程度に，偽の還元動作を追加したものとみなせるので，Knuth 型LR構文解析表を用いるLR(k) 構文解析について述べる．

方法 5.2 の素朴式上昇型解析でみたように，上昇型構文解析では，還元時の左文脈が問題である．この事情を，簡単な次の例で見てみよう．

$$\begin{array}{ll} \mathrm{S'} \to \mathrm{S} & \mathrm{A} \to \mathrm{a} \\ \mathrm{S} \to \mathrm{d\ b\ A} & \mathrm{B} \to \mathrm{a} \\ \mathrm{S} \to \mathrm{e\ c\ B} & \end{array}$$

スタックの中に記号 a をプッシュしたとき，規則 A → a で還元してよいのか規則 B → a で還元してよいのかは，左文脈をすべてみれば区別がつく．すなわち，左文脈の集合が {dba} なら規則 A → a であり，左文脈の集合が {eca} なら規則 B → a である．

さらにもう少し複雑な次の例を見てみよう．

$$\begin{array}{ll} \mathrm{S'} \to \mathrm{S} & \mathrm{S} \to \mathrm{e\ c\ B} \\ \mathrm{S} \to \mathrm{d\ b\ A\ f} & \mathrm{A} \to \mathrm{a} \\ \mathrm{S} \to \mathrm{d\ b\ a\ g} & \mathrm{B} \to \mathrm{a} \end{array}$$

スタックの中に記号 a をプッシュしたとき，左文脈の集合が {eca} なら無条件で規則 B → a による還元である．左文脈の集合が {dba} で先読みが f ならば，規則 A → a による還元であり，左文脈の集合が {dba} で先読みが g ならば，シフトである．そして，スタックの中に記号 g をプッシュしたとき，規則 S → d b a g の還元である．

さらに，例えば，次のような例の場合，左文脈の dba と長さ 1 の先読み g では，区別がつかない．

$$\begin{array}{ll} \mathrm{S'} \to \mathrm{S} & \mathrm{S} \to \mathrm{e\ c\ B} \\ \mathrm{S} \to \mathrm{d\ b\ A\ g} & \mathrm{A} \to \mathrm{a} \\ \mathrm{S} \to \mathrm{d\ b\ a\ g} & \mathrm{B} \to \mathrm{a} \end{array}$$

長さ k の先読みが x で，規則 $A \to \alpha$ を還元するときの左文脈が θ のとき，θx を規則 $A \to \alpha$ の**還元文脈** (reduction context) と呼ぶことにする．実は，どのような文脈自由文法 G の文法規則 $A \to \alpha$ と長さ k の先読みについても，還元文脈の集合 $R(A \to \alpha, x)$ は正規集合になることが知られている．還元文脈の集合 $R(A \to \alpha, x)$ は次のように書き下すことができる．

$$R(A \to \alpha, x) = \left\{ \delta\alpha x \;\middle|\; \begin{array}{l} S'\$^k \overset{*}{\Longrightarrow} \delta Axy \Rightarrow \delta\alpha xy \overset{*}{\Longrightarrow} wxy \text{ は最} \\ \text{右導出，x は長さ } k \text{ の終端記号または右} \\ \text{端記号 \$ の記号列} \end{array} \right\}.$$

与えられた文法 G のそれぞれの文法規則と先読み列の組合せの還元文脈を受理する決定性有限オートマトンを，すべて並列に動作させて構文解析することを考えよう．

方法 6.1 (LR(k) 構文解析の原型)

入力: 入力列の終端記号列 w.

出力: 右解析列またはエラーメッセージ.

手順:

1. 入力列の終端記号列 w に k 個の右端記号 \$ をつなげた列 w$\$^k$ を変数 T に代入する．それぞれの有限オートマトンを開始状態にする．スタックを空にする．
2. どれかの有限オートマトンが受理しているか調べる．
   - (i) 0 番目の規則 $S' \to S$ で先読み $\$^k$ の還元文脈用のオートマトンが受理していたら，正常終了する．
   - (ii) 0 番目以外の規則 $A \to \alpha$ で先読み x の還元文脈用のオートマトンが受理していたら，規則 $A \to \alpha$ の番号を出力する．スタックの内容を $\delta\alpha$x とする．$\delta\alpha$x の α を左辺 A に置き換えた記号列 δAx に，変数 T の内容 (入力列の残り) をつなげたものを，新しい変数 T の内容として，ステップ 1. へ行く．
   - (iii) どれも受理せず，変数 T の内容が空列ならエラー終了する．
   - (iv) それ以外はステップ 3. へ行く．
3. 変数 T の先頭の 1 記号 X を取り，それぞれの有限オートマトンに与え

る．スタックに記号 X をプッシュする．ステップ 2. へ行く．□

この構文解析法で問題になるのは，次の点である．

1. 同じ時刻に 2 つ以上のオートマトンが受理を告げないか?
2. 1 つのオートマトンが受理を告げて，それを無視してステップ 2. とステップ 3. を行っていたら，別のオートマトンが受理を告げないか?

すなわち，還元文脈の言葉で述べると次のようになる．

θx が R(A $\rightarrow$ α, x) の要素で，θxψ が R(B $\rightarrow$ β, y) の要素であるとき，必ず $\psi = \varepsilon$ で，x = y で，A $\rightarrow$ α と B $\rightarrow$ β は同一規則と断定できるかということである．これを保証してくれる文法が LR(k) 文法であり，そのとき，この解析法が LR(k) 解析となる．

実際の LR(k) 構文解析の原理は以下のようになる．生成規則と先読みの組に対する還元文脈の集合が，正規言語であることを利用し，これらすべての和集合を認識する有限オートマトンを 1 つ作る．この有限オートマトンにより，1 つ 1 つの還元文脈の認識を行い，そのつど右辺を左辺に還元していく．LR(k) 条件は解析動作の一意性を保証するので，還元かシフトかエラーかが一意的に区別がつく．

解析動作の一意性だけでは，終了性が保証できないので，ただ 1 回左辺に出現する開始記号に到達したら終了することにする．例えば，自由にふるまう開始記号 S が S $\rightarrow$ S という規則をもっている場合，LR(k) 条件を満たすが，永遠に一意的還元が続いてしまうことになる．

第7章　属 性 文 法

ソフトウェアの意味を形式的に定義する代表的な方法に属性文法がある．まず属性文法の定義を述べ，一括型の属性評価法，差分型の属性評価法，1 パス型の属性評価法の中から基本的と考えられるアルゴリズムを説明する．

7.1　基本的定義

意味を形式的に定義する問題の例として，0 以上の 2 進整数の値を正確に定義する問題を考える．最初に 0 以上の 2 進整数を表す記号列の全体を帰納法を使って定義しよう．

定義 7.1 (0 以上の 2 進整数の記号列)

1. 記号列 0 と記号列 1 は 0 以上の 2 進整数である．
2. 記号列 x が 0 以上の 2 進整数ならば，記号列 x·0 と記号列 x·1 も 0 以上の 2 進整数である．　□

これらの記号列が表す値は，さらに次のように帰納的に定義できる．

定義 7.2 (0 以上の 2 進整数の記号列の値)

1. 記号列 0 の表す値は整数 0 である．記号列 1 の表す値は整数 1 である．
2. 記号列 x が 0 以上の 2 進整数で，その表す値が整数 y ならば，記号列 x·0 の表す値は整数 $2\times y$ で，記号列 x·1 の表す値は整数 $2\times y+1$ である．

□

この定義に基づいて，例えば 2 進記号列 110 の表す値を求めると次のように 6 とわかる．

$$
\begin{aligned}
\text{値}\,(110) &= \text{値}\,(11) \times 2 \\
&= (\text{値}\,(1) \times 2 + 1) \times 2 \\
&= (1 \times 2 + 1) \times 2 \\
&= 6.
\end{aligned}
$$

以上のような帰納法に基づく意味定義を，文脈自由文法の枠組みを用いて行ってみよう．まず記号列の構文規則を文脈自由文法で表してみる．非終端記号としては，< 記号列>という非終端記号を用いることにする．

1) <記号列> → 0
2) <記号列> → 1
3) <記号列> → <記号列> 0
4) <記号列> → <記号列> 1

次に非終端記号 <記号列> の示す終端記号列が表す値を，非終端記号が持つ 1 つの属性 "評価値" の値と考え，<記号列>. 評価値 で表す．こうすると，次のように 0 以上の 2 進整数の値が定義できる．生成規則中の出現箇所を区別するため， <記号列$_0$ > や <記号列$_1$ > で <記号列> の出現位置を区別する．

1) <記号列> → 0
　　<記号列>. 評価値 := 0
2) <記号列> → 1
　　<記号列>. 評価値 := 1
3) <記号列$_0$ > → <記号列$_1$ > 0
　　<記号列$_0$ >. 評価値 := <記号列$_1$ >. 評価値 × 2
4) <記号列$_0$ > → <記号列$_1$ > 1
　　<記号列$_0$ >. 評価値 := <記号列$_1$ >. 評価値 × 2 + 1

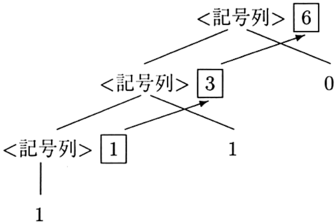


図 7.1 属性計算

実際に与えられた記号列 110 の表わす値を求めるには，図 7.1 のように構文解析木を作成し，その非終端記号の間で，子から親へ向けて，属性の値をバケツリレーすればよい．属性値は四角で囲んで示している．こうすると，最終的に開始記号に評価値 6 が，構文解析木の根の位置にある非終端記号に届き，属性評価が終了する．

構文解析木の上で属性の計算を行う場合，非終端記号は，下位規則に対して生成規則の親としてふるまい，上位規則に対して生成規則の子としてふるまう．したがって，構文解析木中の 1 つの非終端記号に注目した場合，その非終端記号の属性の値が，親としてふるまっている下位の生成規則側から決定すべきか，子としてふるまっている上位の生成規則側から決定すべきかの区別がつかなくなってしまう．

そこで非終端記号の属性を，親としてふるまっている側の生成規則から決定する属性と，子としてふるまっている側の生成規則から決定する属性の 2 種類に分割してしまうことにする．親としてふるまっている側の生成規則から決定する属性を**合成属性**，子としてふるまっている側の生成規則から決定する属性を**相続属性**と呼ぶ．つまり，合成属性の値は下の生成規則から上の生成規則へと決まっていき，相続属性の値は上の生成規則から下の生成規則へと決まっていく．したがって，図 7.2 に示すように 1 つの生成規則に対しては，親の合成属性の値の決め方と子の相続属性の値の決め方を指定しておけばよいことになる．以上のような考え方に基づいて，文脈自由言語の意味を定義する方法が属性文法である．

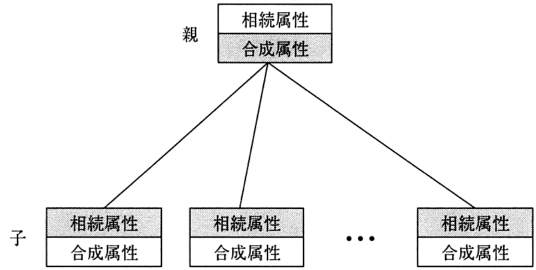


図 7.2

先の 0 以上の 2 進整数の記号列の値を定義する例を，文脈自由文法

$$N \to L,\ L \to L\ B,\ L \to B,\ B \to 1,\ B \to 0$$

と，合成属性 value を用いて記述すると次のようになる．この属性文法による値定義では，規則 B → 1 のレベルで， 4 の位の 1 と，2 の位の 1 の区別はできていないことに注意されたい．

1.	N→L	N.value := L.value
2.	L_0→L_1 B	L_0.value := L_1.value × 2 + B.value
3.	L→B	L.value := B.value
4.	B→1	B.value := 1
5.	B→0	B.value := 0

それでは属性文法の形式的定義を示す．

定義 7.3 (属性文法)
1 つの**属性文法** AG = (G, A, F) は，次の 3 つの構成要素からなる．
1. 文脈自由文法 G

構文を定義するための 1 つの文脈自由文法 G = (N, T, P, S) で，この文脈自由文法を**基底文法**と呼ぶ.

2. 属性集合の集合 A

各非終端記号 X に対する属性集合 A(X) の集合 A(N) = { A(X) : X は N の要素 }. 集合 A(X) は共通要素を持たない 2 つの集合 $A{\uparrow}(X)$ と $A{\downarrow}(X)$ に分割される．つまり，$A(X) = A{\uparrow}(X) \cup A{\downarrow}(X)$ かつ $A{\uparrow}(X) \cap A{\downarrow}(X) = \phi$ である．$A{\uparrow}(X)$ の要素を**合成属性**，$A{\downarrow}(X)$ の要素を**相続属性**と呼ぶ．合成属性は，文法記号が親として使われるときの属性で，相続属性は，文法記号が子として使われるときの属性である.

3. 意味関数の集合の集合 F

各生成規則 p に対する意味関数の集合 F(p) の集合 F(P) = { F(p) : p は P の要素 }. 集合 F(p) は，生成規則 p

$X_0 \rightarrow X_1\ X_2 \cdots X_m$

における

X_0 の各合成属性の値,

X_1，X_2，…，X_m の各相続属性の値

を定義する意味関数の集合である．意味関数の引数としては

X_0，X_1，X_2，…，X_m の合成属性

X_0，X_1，X_2，…，X_m の相続属性

のすべての値を用いることができる．0 引数の関数は定数を定義している. □

例 7.1 (0 以上の 2 進整数の定義)

0 以上の 2 進整数の値を定義する属性文法を示す．属性 scl は，親から子へ決まっていく相続属性であり，属性 val は，子から親へ決まっていく合成属性である．相続属性を用いることにより，規則 B → 1 のレベルで，4 の位の 1 と，2 の位の 1 の区別がついていることに注意されたい.

1.	N→L	L.scl := 0 N.val := L.val
2.	L_0→L_1 B	L_1.scl := L_0.scl + 1 B.scl := L_0.scl L_0.val := L_1.val + B.val
3.	L→B	B.scl := L.scl L.val := B.val
4.	B→1	B.val := $2^{\mathrm{B.scl}}$
5.	B→0	B.val := 0

□

7.2　一括型属性評価法

属性の評価の方法には，一括型と差分型の 2 種類の方法がある．まず与えられた入力列の属性値を，全く無から計算する一括型の属性評価法について述べる．

1 つの属性文法の下で与えられた入力列に対する属性値は，次のように計算することができる．

1. 基底文脈自由文法により入力列の構文解析を行い，入力列に対する導出木を作成する．
2. 導出木および各生成規則の意味規則から属性依存グラフを作成する．
3. 属性依存グラフから属性値の計算を行う．すべての属性の値が定義されれば，入力列に対する属性の値が正しく定義されていたことになる．1 つ以上の属性の値が未定義ならば，属性文法は循環定義であったことになる．

構文解析の方法については第 5 章で述べているので，属性依存グラフの構成法と一括型の属性計算法について説明する．

方法 7.1 (属性依存グラフの構成)
入力: 属性文法とその 1 つの入力列．

出力: 入力列に対する属性依存グラフ.

手順:

1. 各生成規則に対し, 属性文法の意味関数の定義から, 生成規則の属性依存グラフを作成する. すなわち, 生成規則

   $X_0 \rightarrow X_1\ X_2 \cdots X_m$

   に対し, X_0, X_1, $\cdots$, X_m の合成属性と相続属性を 1 つ 1 つの点とし, それぞれの意味関数

   $y_0 := f(y_1, y_2, \cdots, y_n)$,

   y_0 は X_0 の合成属性か X_1, X_2, $\cdots$, X_m の相続属性, y_1, y_2, $\cdots$, y_n は X_0, X_1, X_2, $\cdots$, X_m の合成属性と相続属性,

   に対し, 有向枝は (y_1, y_0), (y_2, y_0), $\cdots$, (y_n, y_0) を持たせる.
2. 構文解析木中の各生成規則に対して, 1. で求めた生成規則の属性依存グラフを貼り付ける. これで構文解析木全体の属性依存グラフが得られたことになる. □

例 7.2 (属性依存グラフ)

例 7.1 の属性文法の下で入力列 110 に対する属性依存グラフを作ってみよう. まず入力列 110 に対する導出木を作ると図 7.3 のようになる. 各生成規則に対応する生成規則の依存グラフを作ると図 7.4 のようになる. 導出木の上に生成規則の依存グラフを貼り合わせることにより, 入力列 110 に対する属性依存グラフが図 7.5 のように得られる. □

属性依存グラフの計算は, 計算ができるところから開始し, それ以上属性の値が新しく求まらなくなるまで繰り返せばよい. すなわち, 属性依存グラフの上流から下流に向けて計算を行っていけばよい. 与えられた属性依存グラフに循環定義がなければすべての値が計算でき, 正常終了する. 循環定義がある場合には, 一部の値が求まらないまま, 終了する.

方法 7.2 (属性依存グラフによる計算)

入力: 属性文法とその 1 つの属性依存グラフ.

出力: 属性値が一部または全部求まった属性依存グラフ.

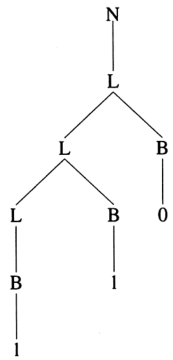


図 7.3　導出木

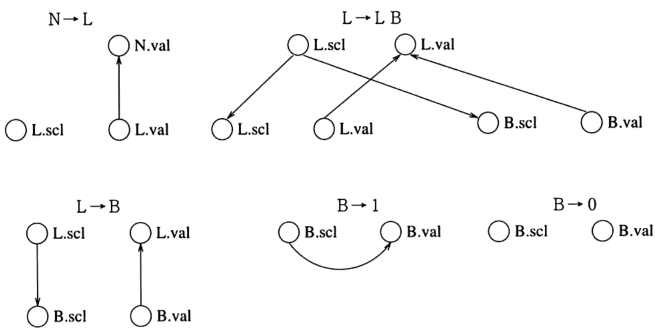


図 7.4　生成規則の属性依存グラフ

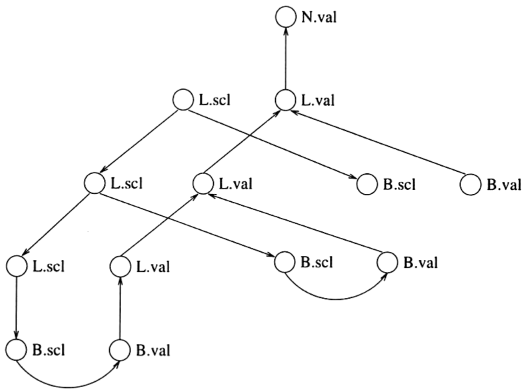


図 **7.5** 属性依存グラフ

手順:

1. 入力枝の個数が 0 個の点が存在するかどうかを調べる．入力枝の個数が 0 個の点が存在するならば，ステップ 2. へ行く．入力枝の個数が 0 個の点が存在しなければ，終了する．このとき，属性の値が未定義の点が 0 個であれば，正常終了となる．未定義の点が 1 個以上あれば，異常終了となる．
2. 入力枝の個数が 0 個の点を 1 つ選び，P と呼ぶ．点 P の属性値を， 0 個以上の計算済みの属性の値と意味関数を用いて計算する．点 P から出ている出力枝をすべて消す．ステップ 1. へ行く． □

例 7.3 (属性計算)

図 7.5 の属性依存グラフの上で属性計算を行ってみよう．図 7.5 の点に順に名前 A ～ M を与え，図 7.6 のように書き直す．入力枝を持たない点は B と G の 2 つであるので, B を選び， B の値 0 を定め，その出力枝を消すと図 7.7 のようになる．入力枝を持たない点は D, F, G であるので，D を選び， D の値を定め，その出力枝を消すと図 7.8 のようになる．この操作を続けていくと最終的に図 7.9 のようになり，すべての属性値が求まる． □

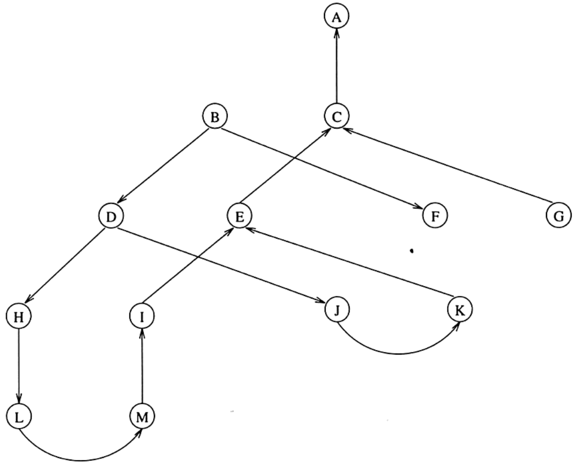


図 7.6　属性計算

与えられた1個の属性依存グラフで循環定義があるかどうかは，方法 7.2 のように区別できるが，無限個または有限個の属性依存グラフのうち少なくとも 1 つの入力列についての属性依存グラフが循環定義となるかどうかの判定は，自明ではない．しかしながら，次のような方針で行うことができる．

（方針）

1. 高さ 1 の部分木の根における属性値の推移的依存関係を，生成規則の属性依存グラフから求める．
2. 高さ n 以下の部分木の根における属性値の推移的依存関係と生成規則の属性依存グラフから，高さ $n+1$ 以下の部分木の根における属性値の推移的依存関係を求める．
3. 高さ 1 以上の部分木の根における属性値の推移的依存関係を出尽くすまで数え上げる．

与えられた属性文法から生成される任意の導出木を T とし，その属性依存グラフを D(T) とする．正しい属性文法であるかどうかは，そのすべての属性依存グラフ D(T) に有向閉路がないかどうかと等価になる．

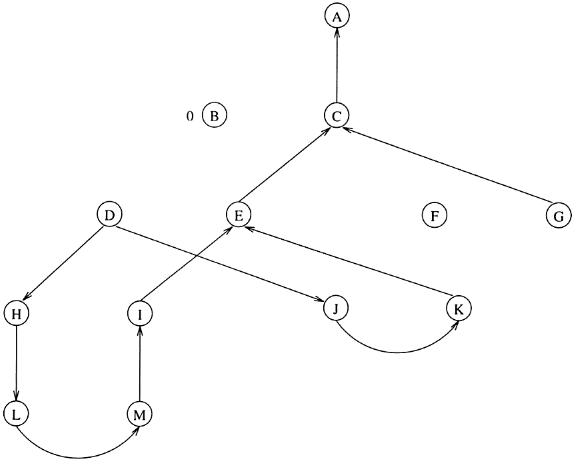


図 **7.7**　属性計算

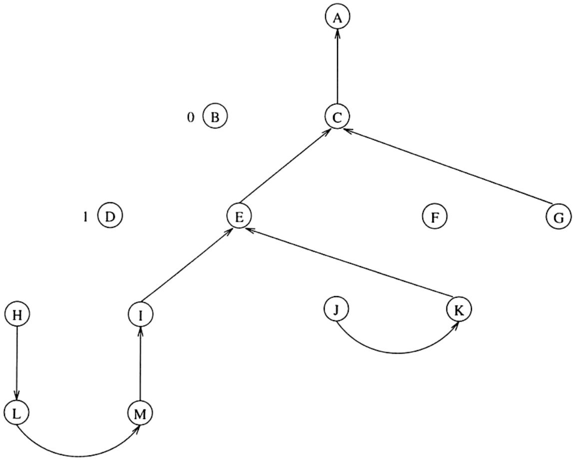


図 **7.8**　属性計算

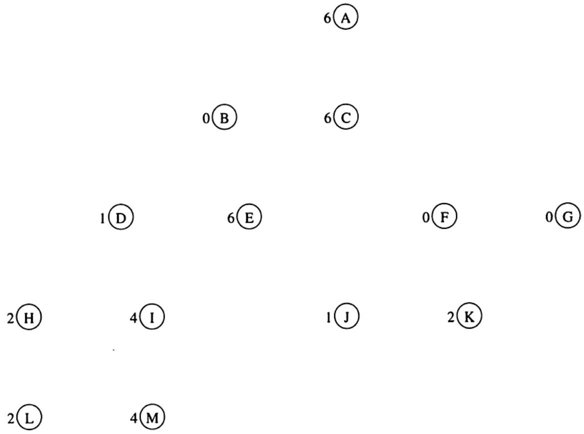


図 **7.9**　属性計算

定義 7.4 (拡大グラフ)
生成規則 p の属性依存グラフを D_p で表す．各 p に対して，その右辺の j 番目の文法記号 X_j，右辺の長さ n に対して，その属性集合を点集合とする有向グラフ G_j が与えられたとする．このとき D_p に，各 G_j に存在する有向枝を追加して得られる有向グラフを $D_p[G_1, G_2, \cdots, G_n]$ で表すことにする． □

方法 7.3 (循環定義の判定)
入力: 属性文法 G.
出力: 属性文法 G が循環定義ならば，“循環定義” と出力，循環定義でなければ，“正しい定義” と出力.
手順:

1. 属性文法 G の各文法記号 X に対し，その属性集合を点集合とする有向グラフの集合 S(X) を対応させる．開始時は，X が終端記号ならば S(X) を空集合とし，X が非終端記号ならば S(X) を X の属性集合が点集合で，有向枝 0 本の有向グラフ 1 つのみとする．
2. 各 S(X) に，それ以上新しくグラフを追加できなくなるまで以下を繰り

返す．

1 つの生成規則を選び，左辺の非終端記号を X_p ，右辺の j 番目の文法記号を X_j，右辺の長さを n とする．各 $S(X_j)$ から，1 つの有向グラフを選び，D'_j とする．有向グラフ $D_p[D'_1, \cdots, D'_n]$ を作り，その中で X_p の属性 a から X_p の属性 b へ推移的依存関係があるとき，$S(X_p)$ に X_p の属性集合を点集合とし，対応する属性 a から b へ有向枝を直接に持つ有向グラフを，既に存在していない場合，$S(X_p)$ に追加する．

3. どんな生成規則 p と関連する S(X) から選んだどんな有向グラフ D'_1, $\cdots$, D'_n に対しても，$D_p[D'_1, \cdots, D'_n]$ に有向閉路が存在しないかどうかを調べる．有向閉路が存在しなければ，“正しい定義” と出力する．有向閉路が存在しなれば，“循環定義” と出力する． □

なお，属性文法の循環定義判定問題は，本質的に指数関数時間かかることが知られている．

例 7.4 (循環定義の判定)

例 7.1 の属性文法の場合で方法 7.3 を実行すると，各文法記号 X に対する有向グラフの集合 S(X) は最初以下のようになる．

$$S(N) = \{ \underset{\text{val}}{\circ} \}, S(L) = \{ \underset{\text{scl}}{\circ} \quad \underset{\text{val}}{\circ} \}, S(B) = \{ \underset{\text{scl}}{\circ} \quad \underset{\text{val}}{\circ} \},$$
$$S(1) = \{\}, S(0) = \{\}$$

そして最終的には次のようになる．

$$S(N) = \{ \underset{\text{val}}{\circ} \}, S(L) = \{ \underset{\text{scl}}{\circ} \quad \underset{\text{val}}{\circ}, \quad \underset{\text{scl}}{\circ} \rightarrow \underset{\text{val}}{\circ} \},$$
$$S(B) = \{ \underset{\text{scl}}{\circ} \quad \underset{\text{val}}{\circ}, \quad \underset{\text{scl}}{\circ} \rightarrow \underset{\text{val}}{\circ} \}, S(1) = \{\}, S(0) = \{\}$$

これらの有向グラフの集合と各生成規則の依存グラフから，どのように組み合わせても有向閉路は存在しないことがわかる． □

7.3 差分型属性評価法

1つの属性値付きの全体木が与えられたときに，その木の1つの節点を更新節点と決め，更新節点の部分木全体を削除し，更新節点に属性値付きの新しい部分木を貼り合わせ，新しい全体木を作ることを考える．なるべく少ない手間でこの新しい全体木の属性の値を正しい属性付きの木に回復させることを，差分型の属性評価という．

新しい全体木の節点のうち，もともとの全体木に属す節点は更新前の古い属性の値を持ち，新しい部分木に属す節点は更新後の新しい属性の値を持っている．ただし，境界線にあたる更新節点においては，相続属性は更新後の新しい属性の値で，合成属性は更新前の古い属性の値とする．この約束により，意味関数の引数と結果の関係が一致せず矛盾している属性の値は，更新節点の属性のみとなる．

これは何故かというと，更新節点以外の相続属性を決める上位の生成規則，合成属性を決める下位の生成規則には一切変更がないこと，更新節点の相続属性は古い上位の生成規則から，更新節点の合成属性は新しい下位の生成規則から決まるためである．

差分型の属性評価を行う上では，以下のような 2 種類の無駄な再計算が行われる可能性がある．

1. ある属性の値が更新前の値と同一にもかかわらず，その属性の値に依存している属性の値を再計算し続けてしまう．
2. 属性値の依存関係で下流の属性の値を先に再計算した後，上流にある属性の値の再計算による変更が発生し，下流の属性の値をもう一度再計算しなければならなくなってしまう．

これらの無駄な再計算を防止するには，属性値の依存関係の下流・上流関係を調べ，更新前の属性の値と更新後の属性の値を比較しながら，少しずつ修正範囲を拡大していくことが必要となる．

定義 7.5 (推移的依存関係)
属性依存グラフにおいて，直接的な依存関係を1つ以上たどって得られる間接的な依存関係を**推移的依存関係**と呼ぶ．この推移的依存関係に 2 つの推

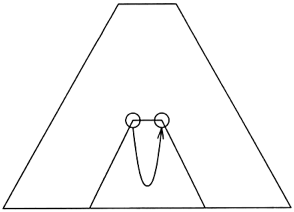


図 **7.10**　下位の推移的依存関係

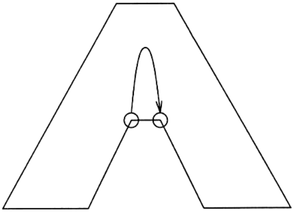


図 **7.11**　上位の推移的依存関係

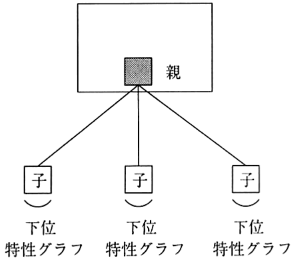


図 **7.12**　下位拡張

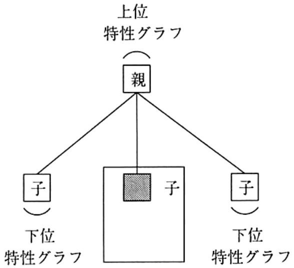


図 **7.13**　上位拡張

移的依存関係を区別する．部分木の根の属性の間の推移的依存関係で，部分木の内側の依存関係のみを用いて得られる関係を，下位の推移的依存関係と呼ぶ．また部分木の根の属性の間の推移的依存関係で，部分木の外側の依存関係のみを用いて得られる関係を，上位の推移的依存関係と呼ぶ．(図 7.10 と図 7.11 にそれぞれの概念図を示す.)　□

定義 7.6 (下位特性グラフ)
部分木の根の属性集合を点集合として，属性 a から属性 b へ下位の推移的依存関係がある場合だけ，属性 a から属性 b へ有向枝を持つ有向グラフを**下位特性グラフ**と呼ぶ.　□

定義 7.7 (上位特性グラフ)
部分木の根の属性集合を点集合として，属性 a から属性 b へ上位の推移的

依存関係がある場合だけ，属性 a から属性 b へ有向枝を持つ有向グラフを**上位特性グラフ**と呼ぶ． □

定義 7.8 (有向グラフの下位拡張)
生成規則 p を $s_0 \to s_1\ s_2 \cdots s_k$ とする．有向グラフの生成規則 p による節点 s_0 の**下位拡張**とは，既に存在する s_0 の属性集合を表す点集合に関して，子の節点 s_1，s_2，$\cdots$，s_k の属性を点に追加し，生成規則 p の属性依存グラフの有向枝，各子の節点 s_1，s_2，$\cdots$，s_k の下位特性グラフの有向枝を追加することである．(概念図を図 7.12 に示す.) □

定義 7.9 (有向グラフの上位拡張)
生成規則 p を $s_0 \to s_1\ s_2 \cdots s_k$ とする．有向グラフの生成規則 p による節点 s_j の**上位拡張**とは，既に存在する s_j の属性集合を表す点集合に，親の節点 s_0，兄弟の節点 s_1，s_2，$\cdots$，s_{j-1}，s_{j+1}，$\cdots$，s_k の属性を点として追加し，生成規則 p の属性依存グラフの有向枝，親の節点 s_0 の上位特性グラフの有向枝，兄弟の節点 s_1，s_2，$\cdots$，s_{j-1}，s_{j+1}，$\cdots$，s_k の下位特性グラフの有向枝を追加することである．(概念図を図 7.13 に示す.) □

以下の方法は，変更の必要な属性値の個数に比例した手間で属性再計算が終了する．

方法 7.4 (差分型属性評価法)
入力: 属性値付きの全体木と更新点と属性値付きの部分木．
出力: 再計算された属性値付きの全体木．
手順:
以下の手続きを属性値付きの新しい全体木とその更新点に関し呼び出す．

PROPAGATE(T : 完全な属性付き木，r : T の 1 つの非終端節点)
begin

1. r の属性集合を点集合とし，r の上位特性グラフと下位特性グラフの有向枝の和集合を有向枝集合とする有向グラフを M とする．
2. r の相続属性の値は継ぎ木した部分木側の値，r の合成属性の値はもともとの全体木の側の値である．
3. S を M の入力枝 0 個の点の集合とする．

4. E を M の点の集合とする．
5. S が空集合でない間以下の処理を繰り返す．
    - 5-1. S から属性インスタンスを 1 つ取り b とし，S から消去する．
    - 5-2. E に b が含まれるなら，
        - 5-2-1. E から b を消去する．
        - 5-2-2. b を b の引数の値を用いて計算する．
        - 5-2-3. b の値が評価前後で異なり，かつ b に直接依存している D(T) の属性インスタンスが M の中にないなら，
            - 5-2-3-1. n を b が付いている非終端節点とする．
            - 5-2-3-2. b に直接依存し，M の中にはない属性インスタンスが，子の属性ならば，M から n の下位特性グラフを取り除いてから M を下位拡張し，親なら，M から n の上位特性グラフを取り除いてから M を上位拡張する．
            - 5-2-3-3. S に，拡張されたノード中で入力枝が 0 個のものを追加する．
        - 5-2-4. M の中の b に直接依存しているそれぞれの属性インスタンス c に対して
            - 5-2-4-1. E に c を入れる．
    - 5-3. M の中の b に直接依存しているそれぞれの属性インスタンス c に対して
        - 5-3-1. M から枝 (b, c) を取り除く．
        - 5-3-2. M における c の入力枝が 0 個ならば，
            - 5-3-2-1. S に c を入れる．

end

□

方法 7.4 で使用した変数 E は，再計算によって値の変化した属性に直接依存している属性を記録するために使用している．これにより，引数が変化した属性のみを再計算することができ，無駄な再計算を避けることができる．

なお，特性グラフの作り方であるが，特性グラフを最初から作る場合には次のようにする．

1. 下位特性グラフは，対象となる部分木のみに依存するので，葉から根へ上昇型で木を訪問して作成する．
2. 上位特性グラフは，各節点の下位特性グラフの情報を用いて，根から葉へ下降型で木を訪問して作成する．つまり，親の上位特性グラフと兄弟の下位特性グラフと生成規則の依存グラフで上位特性グラフが作れる．

また，特性グラフを更新時に作り直すには，以下の特性グラフを更新すればよい．

1. 継ぎ木部分の根の上位特性グラフと下位特性グラフ
2. 継ぎ木部分の根から親をたどって木全体の根まで行くときに通過する節点の上位特性グラフ
3. 上記 2. 以外の節点の下位特性グラフ

例 7.5 (差分型属性評価)

得点の列が与えられたときの合計得点を計算する属性文法で，差分型属性評価法を説明する．ただし，得点の和が負の場合は，和を 0 点として計算することにする．また，最初の持ち点を 0 点とする．

0.	S→L	L.b := 0
1.	L_0→L_1 B	L_1.b := L_0.b B.b := L_1.e L_0.e := B.e
2.	L→B	B.b := L.b L.e := B.e
3.	B→**num**	B.e := **if** B.b + value(**num**) > 0 **then** B.b + value(**num**) **else** 0

例えば，入力列が 3, 2, − 5, − 2, − 3, 2, 4 の場合，得点経過は次のようになり，

入力列		3	2	−5	−2	−3	2	4
得点経過	0	3	5	0	0	0	2	6

得点合計は 6 点となる．この入力に対する属性付き構文解析木は図 7.14 である．さて，終端記号列 32 を葉のレベルに持ち，根が L，高さ 3 の部分木を図 7.15 の新しい部分木で置換してみよう．

部分木の更新点を r とし，r における下位特性グラフと上位特性グラフを

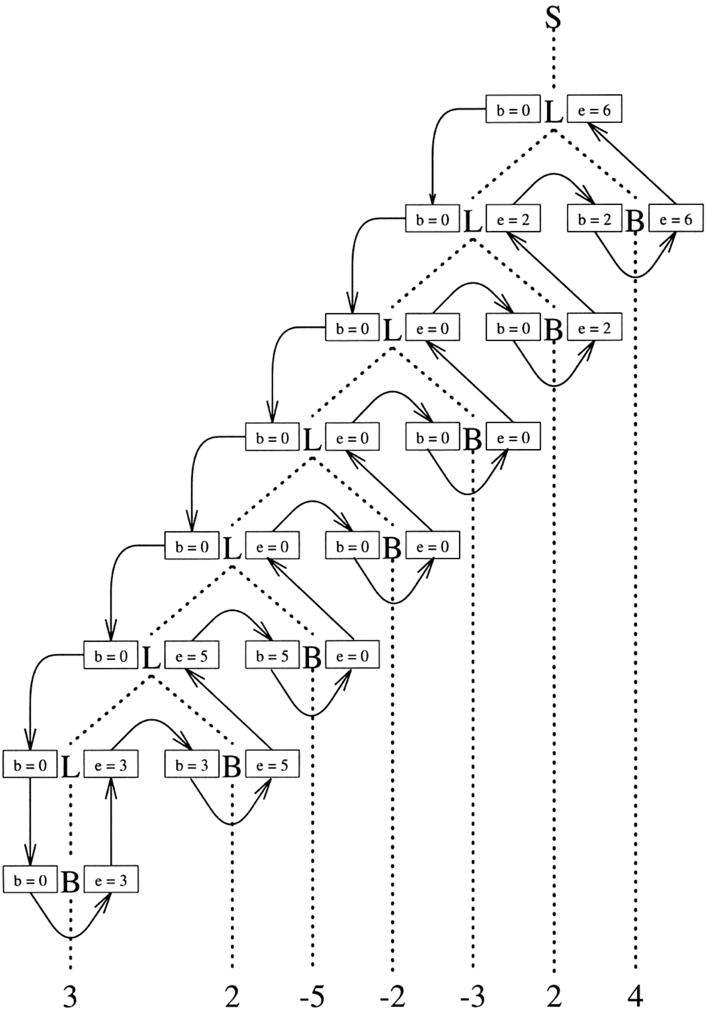


図 **7.14** 属性付き構文解析木

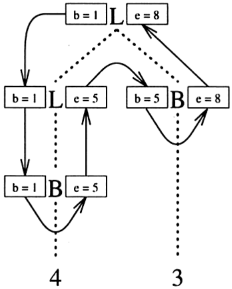


図 7.15　部分木

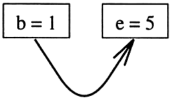


図 7.16　下位特性グラフ

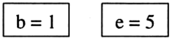


図 7.17　上位特性グラフ

求めると，それぞれ図 7.16，7.17 となる．

更新点 r における属性の値は，この例の場合，相続属性 b の値 1 は新しい部分木側の値で，合成属性 e はもとの全体木の側の値である．したがって，修正範囲 M の初期グラフは図 7.16 となり，再評価は上流にある属性 b から始まる．修正範囲の拡張過程は図 7.19 のようになり，最終的な結果は図 7.18 のようになる．図 7.18 の中で属性を表す点の下についている箱は，変更前の値を示す．変更がない場合は，* を記入している．　□

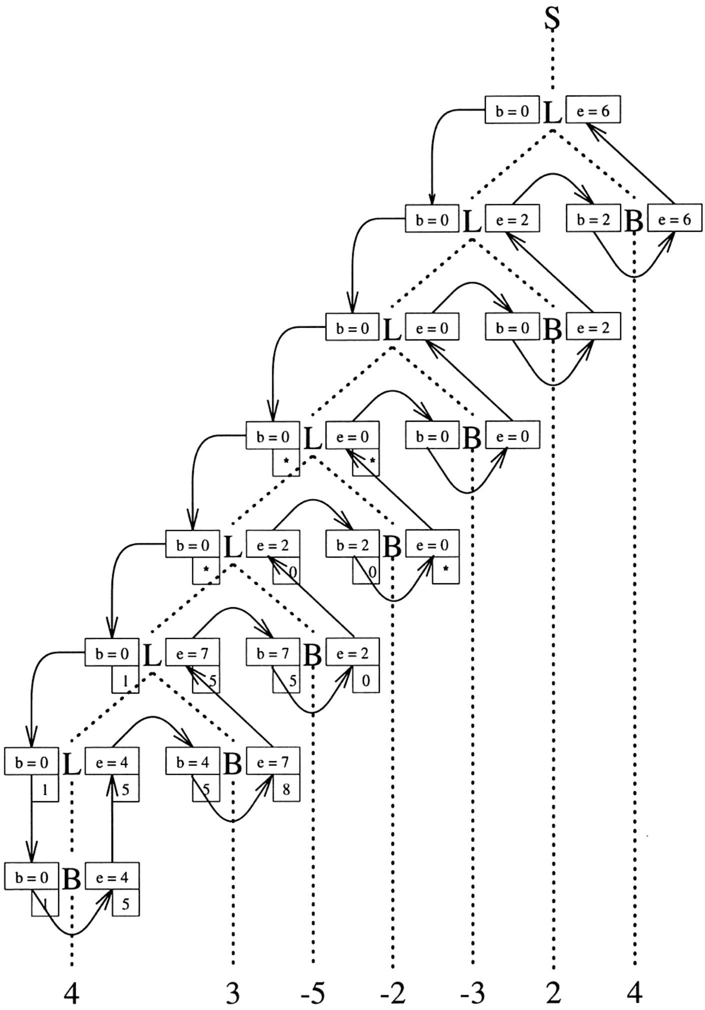

S
b = 0
L
e = 6
b = 0
L
e = 2
b = 2
B
e = 6
b = 0
L
e = 0
b = 0
B
e = 2
b = 0
*
L
e = 0
*
b = 0
B
e = 0
b = 0
*
L
e = 2
0
b = 2
0
B
e = 0
*
b = 0
1
L
e = 7
5
b = 7
5
B
e = 2
0
b = 0
1
L
e = 4
5
b = 4
5
B
e = 7
8
b = 0
1
B
e = 4
5
4
3
-5
-2
-3
2
4

図 7.18　最終結果

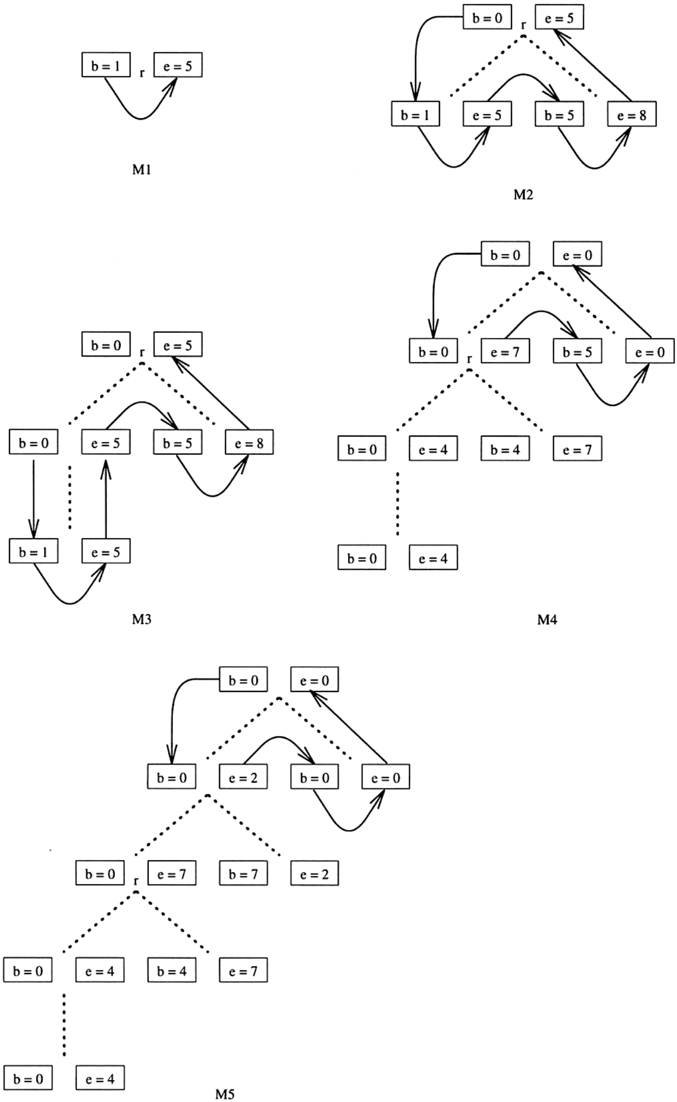


図 7.19　修正範囲の拡張過程

7.4　1パス型属性評価法

上昇型や下降型の構文解析に合わせて，属性依存グラフを作らずに，属性評価を行う方法に，1パス属性評価法がある．以下では，扱える範囲も広く，理解しやすい形式で，上昇型の基本的方法を示す．

上昇型の構文解析の途中段階では，構文解析木の上位部分は完成していない．したがって，上位部分がどのような木構造で，その結果どのような相続属性の計算式になるかは一般には不明である．Jones 達は，一定の文脈区別法の下で，いかなる上位部分に対しても，相続属性の計算式が一意となる属性文法のクラスを特徴付け，これをコンパイラ作成などに利用した．

以下では，相続属性の計算式が必ずしも一意でなくとも，与えられた入力列の下で，可能な相続属性の値が有限種類であればよいというゆるやかな制限で，広範囲の属性文法のクラスを，1パスで評価する方法を示す．

定義 7.10 (L 属性文法)
すべての生成規則 A $\rightarrow$X$_1$ X$_2$ $\cdots$ X$_n$ に対し，次の 2 つの条件を満足するとき，属性文法を **L 属性文法**であるという．

1. 生成規則の右辺の k 番目の文法記号の相続属性の値は，左辺の非終端記号の相続属性の値と右辺の 1 番目から $k-1$ 番目までの合成属性の値のみに依存する．
2. 生成規則の左辺の非終端記号の合成属性の値は，その非終端記号の相続属性の値と右辺の文法記号の合成属性の値のみに依存する．　□

定義 7.11 (属性付き項目の上の関係)
LR(k) 項目 [A $\rightarrow\alpha\cdot\beta$, x] の生成規則 A $\rightarrow$ $\alpha\beta$ の各文法記号に属性の値を保持させたものを，**属性付き項目**と呼ぶ．以下の LR(k) 項目の上の 3 種類の関係を属性付き項目に拡張し，

1. 垂直関係 $\downarrow$ ：[A $\rightarrow\alpha\cdot$Bβ, x] $\downarrow$ [B $\rightarrow\cdot\gamma$, y]
2. 非終端記号の引っ越し：[A $\rightarrow\alpha\cdot$Bβ, x] $\xrightarrow{\mathrm{B}}$ [A $\rightarrow\alpha$B$\cdot\beta$, x]
3. 終端記号の引っ越し：[A $\rightarrow\alpha\cdot$aβ, x] $\xrightarrow{\mathrm{a}}$ [A $\rightarrow\alpha$a$\cdot\beta$, x]

属性付き項目の集合 I に対し，I$\downarrow$，I$\xrightarrow{\mathrm{B}}$，I$\xrightarrow{\mathrm{a}}$ をそれぞれ以下のように

計算する．

1. $I\downarrow$

- 項目規則

  $[A \rightarrow \alpha\cdot B\beta, x]$ が I の要素なら，各 B 規則 $B \rightarrow \gamma$ と $\mathrm{FIRST}_k(\beta x)$ の各要素 y について，$[B \rightarrow \cdot\gamma, y]$ を I につけ加える．ただし，属性値も含めた同一視の下で，既に存在する場合にはつけ加えない．

- 属性規則

  (i) $[A \rightarrow \alpha\cdot B\beta, x]$ の B の相続属性の値を，$[B \rightarrow \cdot\gamma, y]$ の B の相続属性の値とする．

  (ii) $\gamma \neq \varepsilon$ で，γ の最左記号 X が非終端記号ならば，X の相続属性の値を，B の相続属性の値から意味関数を用いて定義する．

  (iii) $\gamma = \varepsilon$ ならば，$[B \rightarrow \cdot\gamma, y]$ の B の合成属性の値を，B の相続属性の値から意味関数を用いて定義する．

2. $I \xrightarrow{B}$

- 項目規則

  $[A \rightarrow \alpha\cdot B\beta, x]$ が I の要素で，$[B \rightarrow \gamma\cdot, y]$ が $\mathrm{GOTO}(I, \gamma)$ の要素ならば，項目 $[A \rightarrow \alpha B\cdot\beta, x]$ を $\mathrm{GOTO}(I, B)$ に追加する．ただし，属性値も含めた同一視の下で，既に存在する場合にはつけ加えない．

- 属性規則

  (i) I の項目 $[A \rightarrow \alpha\cdot B\beta, x]$ の B の相続属性の値と $\mathrm{GOTO}(I, \gamma)$ の項目 $[B \rightarrow \gamma\cdot, y]$ の B の相続属性の値が同一ならば，$\mathrm{GOTO}(I, \gamma)$ の項目 $[B \rightarrow \gamma\cdot, y]$ の B の合成属性の値を，$\mathrm{GOTO}(I, B)$ の項目 $[A \rightarrow \alpha B\cdot\beta, x]$ の B の合成属性の値とする．I の項目 $[A \rightarrow \alpha\cdot B\beta, x]$ の $A\alpha B$ の文法記号の属性の値を，$\mathrm{GOTO}(I, B)$ の $[A \rightarrow \alpha B\cdot\beta, x]$ の対応する文法記号の属性の値とする．

  (ii) さらに $\beta = \varepsilon$ ならば，$[A \rightarrow \alpha B\cdot\beta, x]$ の A の合成属性の値を，A の相続属性と αB の文法記号の合成属性の値から意味関数を用いて定義する．

3. $I \xrightarrow{a}$

- 項目規則

項目 [A →α·aβ, x] がIの要素ならば，項目 [A →αa·β, x] を GOTO(I, a) に付け加える．ただし，属性値も含めた同一視の下で，既に存在する場合にはつけ加えない．

- 属性規則
  (i) I の項目 [A →α·aβ, x] の Aα の文法記号の属性の値を，GOTO(I, a) の項目 [A →αa·β, x] の Aα の文法記号の属性の値とする．
  (ii) さらに $\beta = \varepsilon$ ならば，[A →αa·β, x] の A の合成属性の値を，A の相続属性の値と αa の文法記号の合成属性の値から，意味関数を用いて定義する．□

定義 7.12 (有限条件)
どのような属性付き項目 [A →α · β, x] についても，{[A →α · β, x]}↓∗ が有限集合となるL属性文法を，**有限条件**を満たすL属性文法という．□

以下の1パス属性評価の原理は，次のように説明できる．

1. 項目 [A →α·Bβ, x] が状態 I_1 に存在するとき，これは非終端記号 B の各規則を認識するための認識手続きを， B の相続属性の値を手続きへの入力パラメータとして渡して，呼び出していることを表す．
2. 項目 [B →γ·, y] が状態 I_2 に存在するとき，これは非終端記号 B の1つの生成規則 B →γ の認識が終了し， B の合成属性の値の計算も終了したことを表す．
3. 認識の終了した手続きから呼んでいた手続きへの復帰に際しては，呼び出すときの相続属性の値が一致することを確認する．そして項目 [A →αB·β, x] を状態 GOTO(I_1, B) に作る．属性の値は I_1 の [A →α·Bβ, x] の Aα の文法記号の属性， B の相続属性と I_2 の [B →γ·, y] の B の合成属性となる．

方法 7.5 (1パス評価法)
入力: 基底文法が LR(1) 文法で，しかも有限条件を満たすL属性文法とその入力列．
出力: 属性評価をともなう右解析列．

手順:

1. 解析スタックの中身を空にする．解析スタックにしまう状態は，名前フィールドと項目フィールドを持つ．
2. 開始項目 $[S' \to \cdot S, \varepsilon]$ の S の相続属性の値を計算する．開始状態 I_0 を解析スタックにプッシュする．I_0 の項目集合フィールドに以下の属性付き項目集合を格納する．

   $\{[S' \to \cdot S, \varepsilon]\}\downarrow *$.
3. ステップ 4. を永遠に繰り返す．
4. 解析スタックの先頭の状態を I，入力列の先頭の記号を X とする．入力列が ε のときは，$X = \varepsilon$ とする．

   (i) X = a，ACTION(I, X) = "shift"，GOTO(I, X) = "J" の場合
   入力列の先頭の X を削除し，解析スタックに状態の名前 J をプッシュする．J の項目集合フィールドに以下の属性付き項目集合

   $(I \xrightarrow{a})\downarrow *$

   を格納する．

   (ii) ACTION(I, X) = "reduce p"，p 番目の生成規則の右辺長が m，解析スタックの内容が以下の形で，

$\cdots I_m\ I_{m-1} \cdots I_1\ I$

   p 番目の生成規則の左辺 LHS(p) に対し GOTO(I_m, LHS(p)) = "J" の場合，m 個の状態 I_{m-1}，…，I_1，I を解析スタックからポップする．規則番号 p を出力する．状態 J を解析スタックにプッシュする．J の項目集合フィールドに以下の属性付き項目集合

   $(I_m \xrightarrow{\mathrm{LHS}(p)})\downarrow *$

   を格納する．I_m と I により，この属性付き項目集合が決まる点に注意されたい．
5. ACTION(I, X) = "accept" の場合，accept を出力し，解析を正常終了する．
6. ACTION(I, X) = "error" の場合，error を出力し，解析を正常終了する．

□

例 7.6 (1 パス属性評価)
以下の属性文法を用いて，入力列 .11 を，1 パス属性評価してみる．属性 scl は相続属性で，属性 val と loc は合成属性である．

1.	S→. L	L.scl := −1
		S.val := L.val
2.	L_0→L_1 B	L_1.scl := L_0.scl
		L_0.loc := L_1.loc − 1
		B.scl := L_1.loc − 1
		L_0.val := L_1.val + B.val
3.	L→B	B.scl := L.scl
		L.loc := L.scl
		L.val := B.val
4.	B→1	B.val := 2 ↑B.scl
5.	B→0	B.val := 0

属性評価の様子は図 7.20 のようになる．各項目の属性フィールドは，上から順に scl，val，loc を表している．最終的に属性値 3/4 が求まる．□

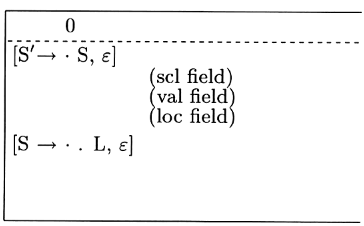


図 7.20 (1)　属性の評価過程

0	1	
$[S' \to \cdot S, \varepsilon]$	$[S \to \cdot.L_{-1}, \varepsilon]$	(scl field) (val field) (loc field)
$[S \to \cdot\ .L, \varepsilon]$	$[L_{-1} \to \cdot L_{-1}\ B, \varepsilon/1/0]$	
	$[L_{-1} \to \cdot B_{-1}, \varepsilon/1/0]$	
	$[B_{-1} \to \cdot 1, \varepsilon/1/0]$	
	$[B_{-1} \to \cdot 0, \varepsilon/1/0]$	

0	1	4
$[S' \to \cdot S, \varepsilon]$	$[S \to \cdot.L_{-1}, \varepsilon]$	$[B_{\substack{-1\\1/2}} \to 1\cdot, \varepsilon/1/0]$
$[S \to \cdot.L, \varepsilon]$	$[L_{-1} \to \cdot L_{-1}\ B, \varepsilon/1/0]$	
	$[L_{-1} \to \cdot B_{-1}, \varepsilon/1/0]$	
	$[B_{-1} \to \cdot 1, \varepsilon/1/0]$	
	$[B_{-1} \to \cdot 0, \varepsilon/1/0]$	

図 7.20 (2)　属性の評価過程

0	1	3
$[S' \to \cdot S, \varepsilon]$	$[S \to .\cdot L_{-1}, \varepsilon]$	$[L_{\substack{-1\\1/2\\-1}} \to B_{\substack{-1\\1/2}} \cdot, \varepsilon/1/0]$
$[S \to \cdot .L, \varepsilon]$	$[L_{-1} \to \cdot L_{-1}\ B, \varepsilon/1/0]$	
	$[L_{-1} \to \cdot B_{-1}, \varepsilon/1/0]$	
	$[B_{-1} \to \cdot 1, \varepsilon/1/0]$	
	$[B_{-1} \to \cdot 0, \varepsilon/1/0]$	

0	1	2
$[S' \to \cdot S, \varepsilon]$	$[S \to .\cdot L_{-1}, \varepsilon]$	$[S_{1/2} \to .L_{\substack{-1\\1/2\\-1}} \cdot, \varepsilon]$
$[S \to \cdot .L, \varepsilon]$	$[L_{-1} \to \cdot L_{-1}\ B, \varepsilon/1/0]$	$[L_{-1} \to L_{\substack{-1\\1/2\\-1}} \cdot B_{-2}, \varepsilon/1/0]$
	$[L_{-1} \to \cdot B_{-1}, \varepsilon/1/0]$	$[B_{-2} \to \cdot 1, \varepsilon/1/0]$
	$[B_{-1} \to \cdot 1, \varepsilon/1/0]$	$[B_{-2} \to \cdot 0, \varepsilon/1/0]$
	$[B_{-1} \to \cdot 0, \varepsilon/1/0]$	

図 7.20 (3) 属性の評価過程

0	1	2	4
$[S' \to \cdot S, \varepsilon]$	$[S \to \cdot.L_{-1}, \varepsilon]$	$[S_{1/2} \to .L_{\substack{-1\\1/2\\-1}} \cdot, \varepsilon]$	$[B_{\substack{-2\\1/4}} \to 1\cdot, \varepsilon/1/0]$
$[S \to \cdot.L, \varepsilon]$	$[L_{-1} \to \cdot L_{-1}\ B, \varepsilon/1/0]$	$[L_{-1} \to L_{\substack{-1\\1/2\\-1}} \cdot B_{-2}, \varepsilon/1/0]$	
	$[L_{-1} \to \cdot B_{-1}, \varepsilon/1/0]$	$[B_{-2} \to \cdot 1, \varepsilon/1/0]$	
	$[B_{-1} \to \cdot 1, \varepsilon/1/0]$	$[B_{-2} \to \cdot 0, \varepsilon/1/0]$	
	$[B_{-1} \to \cdot 0, \varepsilon/1/0]$		

0	1	2	6
$[S' \to \cdot S, \varepsilon]$	$[S \to \cdot.L_{-1}, \varepsilon]$	$[S_{1/2} \to .L_{\substack{-1\\1/2\\-1}} \cdot, \varepsilon]$	$[L_{\substack{-1\\3/4\\-2}} \to L_{\substack{-1\\1/2\\-1}}\ B_{\substack{-2\\1/4}} \cdot, \varepsilon/1/0]$
$[S \to \cdot.L, \varepsilon]$	$[L_{-1} \to \cdot L_{-1}\ B, \varepsilon/1/0]$	$[L_{-1} \to L_{\substack{-1\\1/2\\-1}} \cdot B_{-2}, \varepsilon/1/0]$	
	$[L_{-1} \to \cdot B_{-1}, \varepsilon/1/0]$	$[B_{-2} \to \cdot 1, \varepsilon/1/0]$	
	$[B_{-1} \to \cdot 1, \varepsilon/1/0]$	$[B_{-2} \to \cdot 0, \varepsilon/1/0]$	
	$[B_{-1} \to \cdot 0, \varepsilon/1/0]$		

図 7.20 (4) 属性の評価過程

0	1	2
[S′ → ·S, ε]	[S → .·L , ε] −1	[S →.L ·, ε] −1 3/4 3/4 −2
[S → ·.L, ε]	[L → ·L B, ε/1/0] −1 −1	[L →L ·B , ε/1/0] −1 −1 −3 3/4 −2
	[L → ·B , ε/1/0] −1 −1	[B → ·1, ε/1/0] −3
	[B → ·1, ε/1/0] −1	[B → ·0, ε/1/0] −3
	[B → ·0, ε/1/0] −1	

0	7
[S′ →·S, ε]	[S′ → S ·, ε] 3/4
[S →·.L, ε]	

図 7.20 (5)　属性の評価過程

あとがき

本書をまとめるにあたり，著者が用いた参考文献や読者が今後さらに関連分野を勉強する際に役立つと思われる参考文献を，本セクションの後に参考文献リストとしてまとめておく．一応の目安として，ジャンル別に列挙するが，本書の索引制作と同様にコンピュータを用いたため，あまり正確な分類ではない．

本書の紙面制作は，LaTeX システムを用いて行われたが，紙面制作の都合で，一部の図は本文に収録しきれなかった．これらのうち，言語処理の方法論に基づいて実際に設計されたソフトウェア生成系の利用例の図をいくつかまとめて示しておきたい．図 1 は AT&T のコンパイラ生成系である YACC システムによる電卓プログラムの定義の例，図 2 は出来上がった電卓プログラムの実行例である．図 3 は Cornell 大学の構造エディタ生成系である CSG システムによる電卓の定義の例，図 4 は同じく出来上がった電卓プログラムの実行例である．YACC システムでは意味定義に通常のプログラムを使うのに対し， CSG システムでは属性文法を用いている．図 5 は Wisconsin 大学の属性文法に基づくプロトコル定義システムである RTAG システムによるデータリンクレイヤのプロトコルの定義例，図 6 は東京工業大学のグラフ構造上の情報処理を行うソフトウェアの生成系である TAHG システムによって生成された論理回路エディタの実行例である．

なお最後に，本書の校正作業は多くの人々の協力を得て行われたが，除去しきれなかった部分の誤植やエラーについては，著者として深くお詫びするとともに，今後速やかに改めたいと思う．

```
%token NUMBER
%left '+' '-'
%left '*' '/'
%%
list:
        | list '\n'
        | list exp '\n' { printf("\tVALUE = %d\n", $2); }
        ;
exp:      NUMBER          { $$ = $1; }
        | exp '+' exp     { $$ = $1 + $3; }
        | exp '-' exp     { $$ = $1 - $3; }
        | exp '*' exp     { $$ = $1 * $3; }
        | exp '/' exp     { $$ = $1 / $3; }
        | '(' exp ')'     { $$ = $2; }
        ;
%%
#include "lex.yy.c"
```

図 1　YACC による電卓の定義

```
%dc
((2 + 5) * (4 - 2))
          VALUE = 14
2 + 5
          VALUE = 7
4 - 2
          VALUE = 2
7 * 2
          VALUE = 14
█
```

図 2　YACC による電卓の実行例

```
root cal;
list cal;
cal: CalL(exp cal)
   | CalN()
   ;
exp:  Nul()
   |  Sum, Dif, Pro, Quo(exp exp)
   |  Con(INT)
   ;

exp {synthesized INT v;};
exp:  Nul{ exp.v = 0; }
   |  Sum{ exp$1.v = exp$2.v + exp$3.v; }
   |  Dif{ exp$1.v = exp$2.v - exp$3.v; }
   |  Pro{ exp$1.v = exp$2.v * exp$3.v; }
   |  Quo{ local STR err;
        err     = (exp$3.v == 0) ? "<--Division by zero--> " : "";
        exp$1.v = (exp$3.v == 0) ? exp$2.v : (exp$2.v / exp$3.v);
        }
   |  Con{ exp$1.v = INT; }
   ;
```

図 3　CSG による電卓の定義

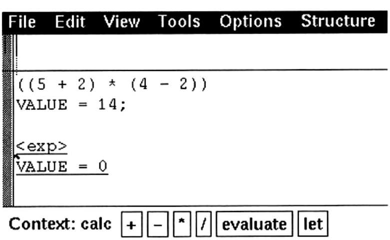


図 4　CSG による電卓の実行例

```
<init>: [U->DT] [N->DT] <loop> .  /* initialize */
      $0.NextFrame = 0
      $0.FrameExp = 0
      $0.Addr = $1.addr
      $0.Port = $1.port
      $2.addr = $1.addr
      $2.port = $1.port
      $2.seq = $0.NextFrame
      $2.ack = 1 - $0.FrameExp
      $2.info = $1.info
;

<loop>: [N->DT] <pass> <ack> <process> <loop> .
      $2.seqnum = $1.seq
      $3.acknum = $1.ack
      <init>.Buffer = $1.info
      | /timer/ <process> <loop> .  /* timeout; retransmit */
      $1.interval = TIMEOUT
;

<pass>: [U<-DT] .  /* new frame for user? */
      if $0.seqnum == <init>.FrameExp
      $1.info = <init>,Buffer
      <init>.FrameExp = 1 - <init>.FrameExp  /* toggle bit */
      | .       /* no new frame */
;

<ack>:  .       /* increment frame after get */
      if $0.acknum == <init>.NextFrame
      <init>.Buffer = #getdata#()  /* blocking read */
      <init>.NextFrame = 1 - <init>.NextFrame
      | .
;
```

図 5 (1)　RTAG の定義例

```
<process>: [N<-DT] .  /* transmit or retransmit frame */
      $1.addr = <init>.Addr
      $1.port = <init>.Port
      $1.info = <init>.Buffer
      $1.seq = <init>.NextFrame
      $1.ack = 1 - <init>.FrameExp   /* flip the bit */
```

図 5 (2)　RTAG の定義例 (続き)

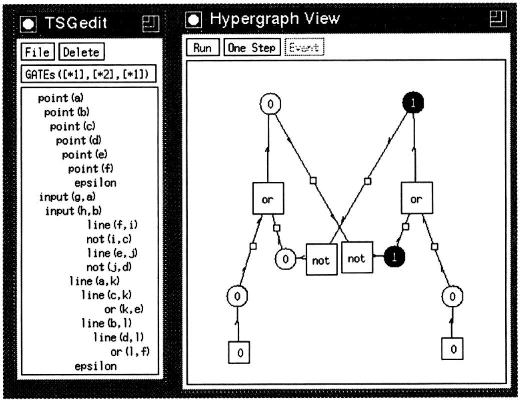


図 6　TAHG による論理回路の実行例

参 考 文 献

形式言語とオートマトンについての参考文献

1) Aho, A. V. and Ullman, J. D.: *The Theory of Parsing, Translation, and Compiling Vol. I: Parsing*, Prentice-Hall, 1972.
2) Aho, A. V. and Ullman, J. D.: *The Theory of Parsing, Translation, and Compiling Vol. II: Compiling*, Prentice-Hall, 1973.
3) Brzozowski, J. A.: Derivatives of Regular Expressions, *J. ACM*, 11(4): 481–494, 1964.
4) Hopcroft, J. E. and Ullman, J. D.: *Formal Languages and Their Relation to Automata*, Addison-Wesley, 1967.
5) Hopcroft, J. E. and Ullman, J. D.: *Introduction to Automata Theory, Language, and Computation*, Addison-Wesley, 1979.
6) Rayward-Smith. V. J.: *A First Course in Formal Language Theory*, Blackwell Scientific Publications, 1983.
7) Salomaa, A.: *Theory of Automata*, Pergamom, 1969.
8) Shannon, C. E. and McCathy, J. (eds.): *Automata Studies*, Princeton University Press, 1956.
9) von Neumann, J.: *Theory of Self-Reproducing Automata*, The University of Illinois Press, 1966.

自然言語についての参考文献

1) Chomsky, N.: On Certain Formal Properties of Grammars, *Information and Control*, 2(2):137–167, 1959.
2) Chomsky, N.: *Aspects of the Theory of Syntax*, MIT Press, 1965.
3) Matsumoto, Y., Kiyono, M., and Tanaka, H.: Facilities of BUP Parsing System, In *Proc. of 1st International Workshop on Natural Language Understanding and Logic Programming*, 1984.
4) Tomita, M.: An Efficient Context-Free Parsing Algorithm for Natural Languages and Its Applications, *Carnegie-Mellon University*, CMU–CS–85–134, 1985.
5) Winograd, T.: *Language as A Cognitive Process, Vol. 1: Syntax*, Addison-Wesley, 1983.

コンパイラについての参考文献

1) Aho, A. V., Sethi, R., and Ullman, J. D.: *Compilers, Principles, Techniques and Tools*, Addison-Wesley, 1986.
2) Aho, A. V. and Ullman, J. D.: *Principles of Compiler Design*, Addison-Wesley, 1977.
3) Gries, D.: *Compiler Construction for Digital Computers*, Addison-Wesley, 1986.
4) Lewis, P. M., Rosenkrantz, D. J., and Stearns, R. E.: *Compiler Design Theory*, Addison-Wesley, 1976.
5) Wirth, N.: *Algorithms + Data Structures = Programs*, Prentice-Hall, 1976.

下降構文解析についての参考文献

1) Conway, M. E.: Design of A Separable Transition-Diagram Compiler, *Comm. ACM*, 6(7):396–408, 1963.
2) Lewis, P. M. and Stearns, R. E.: Syntax-Directed Transduction, *J. ACM*, 15(3):465–488, 1968.

3) 徳田雄洋: 構文解析, 昭晃堂, 1989.

4) Wood, D.: A Note on Top-Down Deterministic Languages, *BIT*, 9(4): 387–399, 1969.

5) Woods, W. A.: Transition Network Grammars for Natural Language Analysis, *Comm. ACM*, 13(10):591–606, 1970.

LR 構文解析についての参考文献

1) Aho, A. V., Johnson, S. C., and Ullman, J. D.: Deterministic Parsing of Ambiguous Grammars, *Comm. ACM*, 18(8):441–452, 1975.

2) DeRemer, F. L.: Simple LR(k) Grammars, *Comm. ACM*, 14(7):453–460, 1971.

3) DeRemer, F. L. and Pennello, T.: Efficient Computation of LALR(1) Look-Ahead Sets, *ACM Trans. Prog. Lang. Syst.*, 4(4):615–649, 1982.

4) Knuth, D. E.: On The Translation of Languages from Left to Right, *Information and Control*, 8(6):607–639, 1965.

5) LaLonde, W. R., Lee, E. S., and Horning, J. J.: An LALR(k) Parser Generators, In *Proc. 3rd IFIP Congress 71*, pages 153–157, 1971.

6) Tokuda, T.: Eliminating Unit Reductions from LR(k) Parsers Using Minimum Contexts, *Acta Informatica*, 15(4):447–470, 1981.

一般文脈自由文法の構文解析についての参考文献

1) Younger, D. H.: Recognition and Parsing of Context-Free Languages in Time n^3, *Information and Control*, 10(2):189–208, 1967.

2) Kasami, T.: An Efficient Recognition and Syntax Analysis Algorithm for Context-Free Languages, AFCRL–65–758, Air Force Cambridge Research Laboratory, Bedford, Mass., 1965.

3) Earley, J.: An Efficient Context-Free Parsing Algorithm, *Comm. ACM*, 13(2):94–102, 1970.

順位構文解析についての参考文献

1) Colmerauer, A.: Total Precedence Relations, *J. ACM*, 17(1):14–30, 1970.
2) McKeeman, W. M., Horning, J. J., and Wortman, D. B.: *A Compiler Generator*, Prentice-Hall, 1970.
3) Inoue, K.: Right Precedence Grammars, *Information Processing in Japan*, 11:24–29, 1971.
4) Wirth, N. and Weber, H.: Euler: A Generation of Algol and Its Formal Definition: Part I, *Comm. ACM*, 9(1):13–23, 1966.

計算モデルと形式的意味論についての参考文献

1) Floyd, R. W.: Nondeterministic Algorithm, *J. ACM*, 14(4):636–644, 1967.
2) Katayama, T. and Sasaki, H.: Global Storage Allocation in Attribute Evaluation, In *Proc. of 13th ACM Symp. on POPL*, pages 26–37, 1986.
3) Kowalski, R.: Algorithm = Logic + Control, *Comm. ACM*, 24(2):147–155, 1979.
4) Marcotty, M., Ledgard, H. F., and Bochmann, G. V.: A Sampler of Formal Definitions, *ACM Computing Surveys*, 8(2):189–276, 1976.
5) Van Wijngaarden, A. et al: Revised Report on the Algorithmic Language ALGOL 68, *Acta Informatica*, 5:1–236, 1974.

属性文法についての参考文献

1) Bochmann, G. V.: Semantic Evaluation from Left to Right, *Comm. of the ACM*, 19(2):55–62, 1976.
2) Katayama, T.: Translation of Attribute Grammars into Procedures, *ACM Trans. Prog. Lang. Syst.*, 6(3):345–369, 1984.
3) Knuth, D. E.: Semantics of Context-Free Languages, *Mathematical Systems Theory*, 2(2):127–145, 1968. Correction : 5(1):95–96, 1971.

差分型属性評価法についての参考文献

1) Reps, T., Teitelbaum, T., and Demers, A.: Incremental Context Dependent Analysis for Language-Based Editors, *ACM Trans. Program. Lang. Syst.*, 5(3):449–477, 1983.
2) Teitelbaum, T. and Reps, T.: The Cornell Program Synthesizer: A Syntax-Directed Programming Environment, *Comm. of the ACM*, 24(9):563–573, 1981.

1パス型属性評価法についての参考文献

1) Jones, N. D. and Madsen, C. M.: Attribute-Influenced LR Parsing, In *Lecture Notes in Computer Science*, 94, pages 393–407, 1980.
2) Nakata, I. and Sassa, M.: L-attributed LL(1) Grammars are LR(1)-attributed, *Information Processing Letters*, 23(6):325–328, 1986.
3) op den Akker, R., Melichar, B., and Tarhio, J.: The Hierarchy of LR-Attributed Grammars, In *Lecture Notes in Computer Science*, 461, pages 13–28, 1990.
4) Sassa, M., Ishizuka, H., and Nakata, I.: ECLR-Attributed Grammars: A Practical Class of LR-Attributed Grammars, *Information Processing Letters*, 24(1):31–41, 1987.
5) Tokuda, T. and Watanabe, Y.: An Attribute Evaluation of Context-Free Languages, *Information Processing Letters*, 52(2):91–98, 1994.
6) Tokuda, T. and Watanabe, Y.: An Efficient Semantic Evaluator for Warped LC(1) Attributed Grammars, *Information Processing Letters*, 53(5):269–276, 1995.

ソフトウェア生成系についての参考文献

1) Gondow, K., Imaizumi, T., Shinoda, Y., and Katayama, T.: Change Management and Consistency Maintenance in Software Development Environments Using Object-Oriented Attribute Grammars, In *Lecture Notes in Computer Science*, 742, pages 77–94, 1993.

2) Habermann, A. N. and Notkin, D.: Gandalf: Software Development Environment, *IEEE Trans. Softw. Eng.*, SE-12(12):1117-1127, 1986.

3) Johnson, S. C.: *YACC — Yet Another Compiler-Compiler*, UNIX Programmer's Manual, AT&T Bell Laboratories, 1978.

4) Kastens, U., Hutt, B., and Zimmermann, E.: GAG: A Practical Compiler Generator, *Lecture Notes in Computer Science*, 141, 1982.

5) Lesk, M. E. and Schmidt, E.: *LEX — A Lexical Analyzer Generator*, UNIX Programmer's Manual, AT&T Bell Laboratories, 1978.

6) Reps, T. and Teitelbaum, T.: *The Synthesizer Generator: A System for Constructing Language-Based Editors*, Springer-Verlag, 1988.

7) Sassa, M., Ishizuka, H., and Nakata, I.: Rie, A Compiler Generator Based on A One-pass-type Attribute Grammar, *Software — Practice and Experience*, 25(3):229–250, 1995.

8) Tokuda, T. and Watanabe, Y.: An Attribute Grammar Modelling of Interactive Figures, *Information Modelling and Knowledge Bases V*: 212–226, 1994.

制約充足法についての参考文献

1) Freeman-Benson, B. N., Maloney, J. H., and Bornig, A.: An Incremental Constraint Solver, *Comm. of the ACM*, 33(1):54–63, 1976.

2) Leler, W.: *Constraint Programming Languages: Their Specification and Generation*, Addison-Wesley, 1988.

3) Maloney, J. H.: Using Constraints for User Interface Construction, *Dept. of Comp. Sci. and Eng., University of Washington*, Tech. Rep. 91–08–12, 1991.

4) Sannella, J. M.: Constraint Satisfaction and Debugging for Interactive User Interface, *Dept. of Comp. Sci. and Eng., University of Washington*, Tech. Rep. 94–09–10, 1994.

索　　引

著者略歴

徳田（とくだ）　雄洋（たけひろ）

1951年　東京都に生まれる
1977年　東京工業大学大学院理工学研究科博士課程情報科学専攻中退
現　在　東京工業大学大学院情報理工学研究科計算工学専攻 助教授
　　　　理学博士
著　書　コンパイラ（共訳，エイホ・ウルマン著，培風館）
　　　　構文解析（昭晃堂）
　　　　はじめて出会うコンピュータ科学・全8巻（岩波書店）

情報数学講座（全15巻）第5巻
言語と構文解析

1995年10月20日　初版1刷発行

検印廃止
NDC 007
ISBN4-320-02655-1

発行者　南條光章

発行所　**共立出版株式会社**
東京都文京区小日向4-6-19
電話　03-3947-2511（代表）
郵便番号112／振替口座　00110-2-57035

印　刷　啓文堂

製　本　関山製本

社団法人
自然科学書協会
会員

Printed in Japan